

## NODE JS

Node.js is an extremely **powerful JavaScript-based platform** used to develop **online chat applications, video streaming sites, single-page applications**, and many other I/O-intensive web applications and web apps. Built on the JavaScript V8 Engine of Google Chrome, It is used by large, established companies and newly-minted startups alike (Netflix, Paypal, NASA, and Walmart, to name a few).

Node.js is **open-source and completely free**, used by thousands of developers around the world. It brings plenty of advantages to the table, making it a better choice than other server-side platforms like Java or PHP.

### What is Node.js?

Node.js is an open-source, cross-platform JavaScript runtime environment and library for running web applications outside the client's browser. Ryan Dahl developed it in 2009, and its latest iteration, version 16.17, was released in Sep 2022. Developers use Node.js to create server-side web applications, and it is perfect for data-intensive applications since it uses an asynchronous, event-driven model.



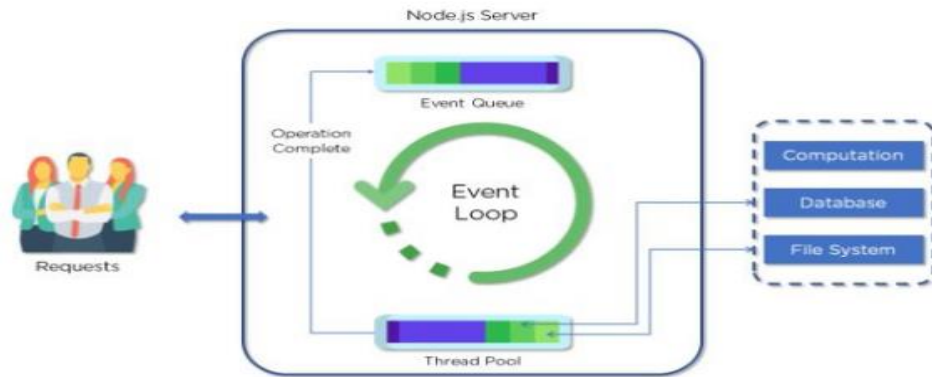
### Why Node.js?

Node.js has become the de facto tool for developing server-side and network applications.

1. **Node.js is really fast:** Having been built on Google Chrome's V8 JavaScript engine, its library is extremely fast for code execution.
2. **Node Package Manager (NPM):** Node Package Manager has more than 50,000 bundles, so whatever functionality is required for an application can be easily imported from NPM.
3. **Node.js uses asynchronous programming:** All APIs of Node.js library are asynchronous (i.e., non-blocking), so a Node.js-based server does not wait for the API to return data. The server calls the API, and in the event that no data is returned, the server moves to the next API the Events module of Node.js helps the server get a response from the previous API call. This also helps with the speed of Node.js.
4. **No buffering:** Node.js dramatically reduces the processing time while uploading audio and video files. Node.js applications never buffer data and simply output the data in chunks.
5. **Single-threaded:** Node.js makes use of a single-threaded model with event looping. As a result, it can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
6. **Highly scalable:** Node.js server responds in a non-blocking way, making it highly scalable in contrast with traditional servers, which create limited threads to handle requests.

## Node.JS Architecture

Node.js uses the "Single Threaded Event Loop" architecture to handle multiple concurrent clients. The Node.js processing model is based on the JavaScript event-based model, along with the JavaScript callback mechanism.

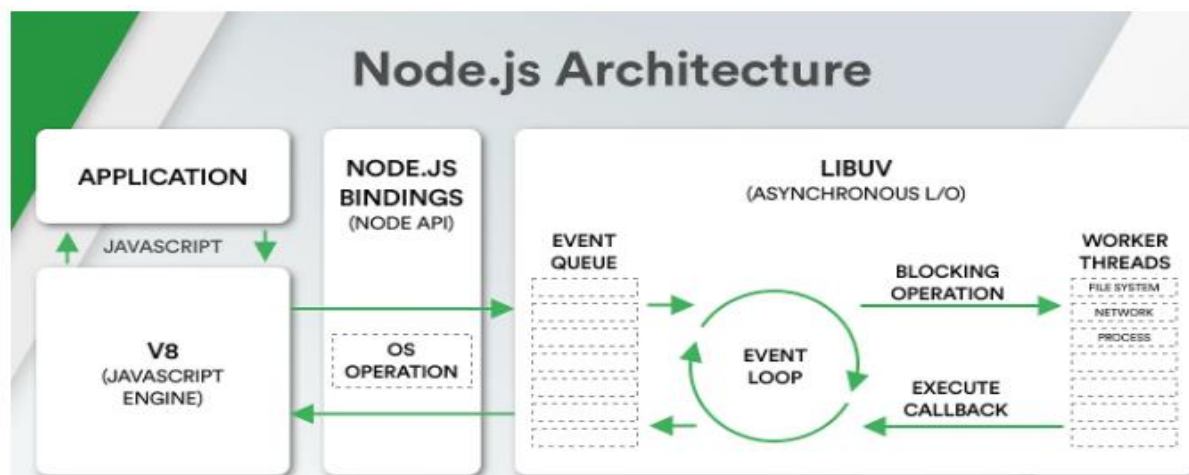


- Clients send requests to Web Server.

Requests can be:

- Querying for data
- Deleting data
- Updating the data, etc.
- Node.js adds the requests to the Event Queue
- Event Loop checks if the requests are simple enough not to require any external resources
- Event Loop processes simple requests and returns the responses to the corresponding clients
- A single thread from Thread Pool is assigned to a single complex request
- Thread Pool performs the required task and returns the response to Event Loop, which in turn, returns the response to the client

Node.js operates on a single-thread, allowing it to handle thousands of simultaneous event loops. Here's a diagram, provided by [Sinform.com](http://Sinform.com), that best illustrates Node.js architecture.



## Parts of Node.JS

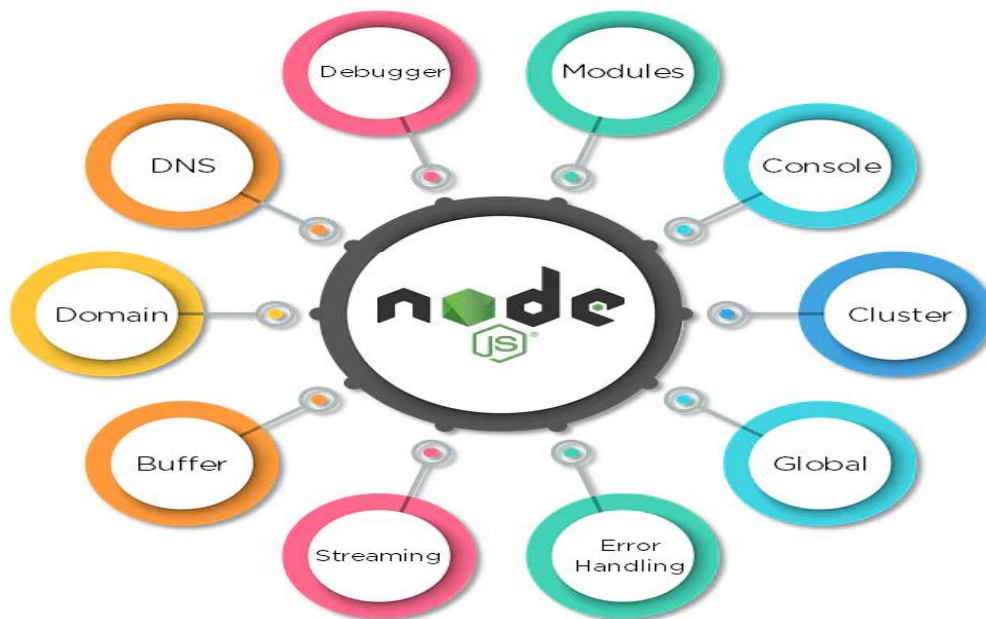


Fig: Parts of Node.js

## Modules

Modules are like JavaScript libraries that can be used in a Node.js application to include a set of functions. In order to include a module in a Node.js application, use the `require()` function with the parenthesis containing the name of the module.

```
// CREATING A WEB SERVER  
  
// Include modules  
var http = require('http');  
var server =  
http.createServer(function(req, res){  
  //write your code here  
});  
server.listen(2000);
```

Fig: Include a module in Node.js

Node.js has many modules that provide the basic functionality needed for a web application. Some of them are mentioned in this table:

Core Modules	Description
http	Includes classes, methods and events to create Node.js http server
util	Includes utility functions useful for developers
fs	Includes events, classes, and methods to deal with file I/O operations
url	Includes methods for URL parsing
querystring	Includes methods to work with query string
stream	Includes methods to handle streaming data
zlib	Includes methods to compress or decompress files

Fig: Node.js modules table

## Node.js HTTP module

Node.js has a built-in module called HTTP. This module enables Node.js to transfer data over the internet. We use the `require()` method to include the HTTP module in an application.

Now, let's create a simple Web server using the HTTP module:

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

http.createServer(function (req, res) {
  res.write('Hello World!');
  res.end();
}).listen(8080);
```

Fig: Web server using HTTP module

- We use the `require()` method to include the HTTP module in the application
- Then, we create a server object using the `createServer` method
- After that, we write a response 'Hello World!' to the client and then end it
- We set the web server to listen at port - 8080

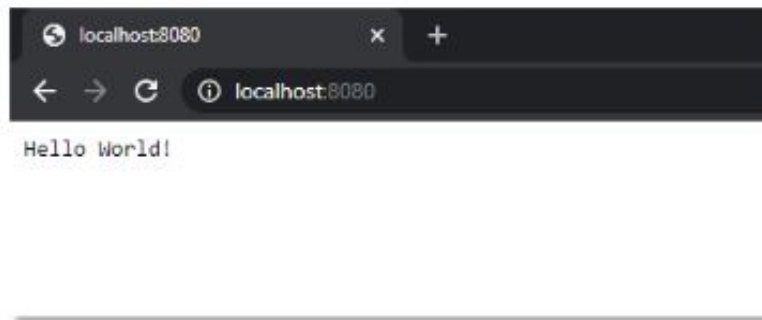


Fig: Web server output

This is what the output looks like on the web browser when we go to the URL with the correct port.

## Node.js File System

Next in the getting started with node.js tutorial we will look at node.js file system. The Node.js file system module enables the file system to work on a computer. We use the require() method to include the file system module in the web application.

**var http=require('fs');**

## Node.js Events

Every action on a computer is considered an event. For example, opening a file, connecting to the internet, etc. Node.js has a built-in events module, where users can create, trigger, and listen for events.

Let's look at a basic implementation of the Events module in a Node.js application:

```
var events = require('events');
var EventEmitter = new events.EventEmitter();

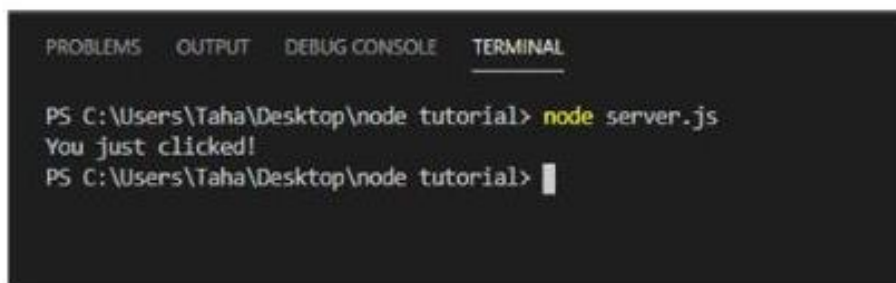
var myEventHandler = function () {
  console.log('You just clicked!');
}

eventEmitter.on('click', myEventHandler);

eventEmitter.emit('click');
```

Fig: Node.js Events implementation

- We use the require() method to include events in our application
- Then, we create an EventEmitter object, which is a module that facilitates interaction between objects in Node.js
- After that, we create an event handler and assign it to the event 'click'
- Finally, we emit/trigger the event



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS C:\Users\Taha\Desktop\node tutorial> node server.js
You just clicked!
PS C:\Users\Taha\Desktop\node tutorial> |
```

Fig: Node.js Events output

The 'myEventHandler' method is called, and the console shows the output when you trigger the event using the 'emit()' method.

## Console

The console is a module that provides a method for debugging that is similar to the basic JavaScript console provided by internet browsers. It prints messages to stdout and stderr.

```
// WRITING "hello world" to console
console.log('hello world');
```

Fig: Node.js console

## Cluster

Node.js is built-on on the concept of single-threaded programming. Cluster is a module that allows multi-threading by creating child processes that share the same server port and run simultaneously.

A cluster can be added to an application in the following way:

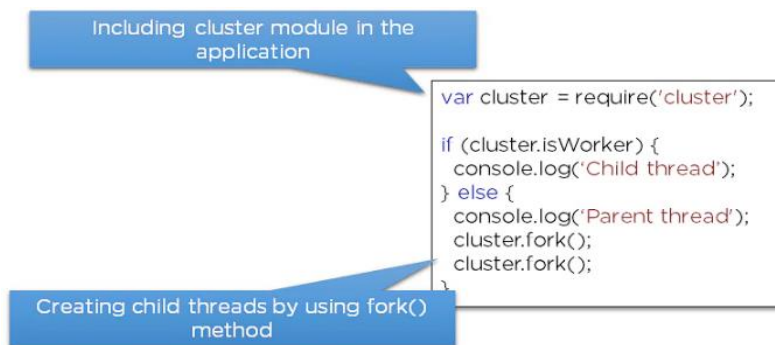


Fig: Add cluster in Node.js

## Global

Global objects in Node.js are available in all modules. These objects are functions, modules, strings, etc. Some Node.js global objects are mentioned in the table below:

Global Objects	Description
__dirname	Specifies the name of the directory that contains the code of application
__filename	Specifies the filename of the code
exports	A reference to the module.exports, shorter to type
module	A reference to the current module
require	Used to import modules, local files, and also JSON

Fig: Global objects table



## Error Handling

Node.js applications experience four types of errors.

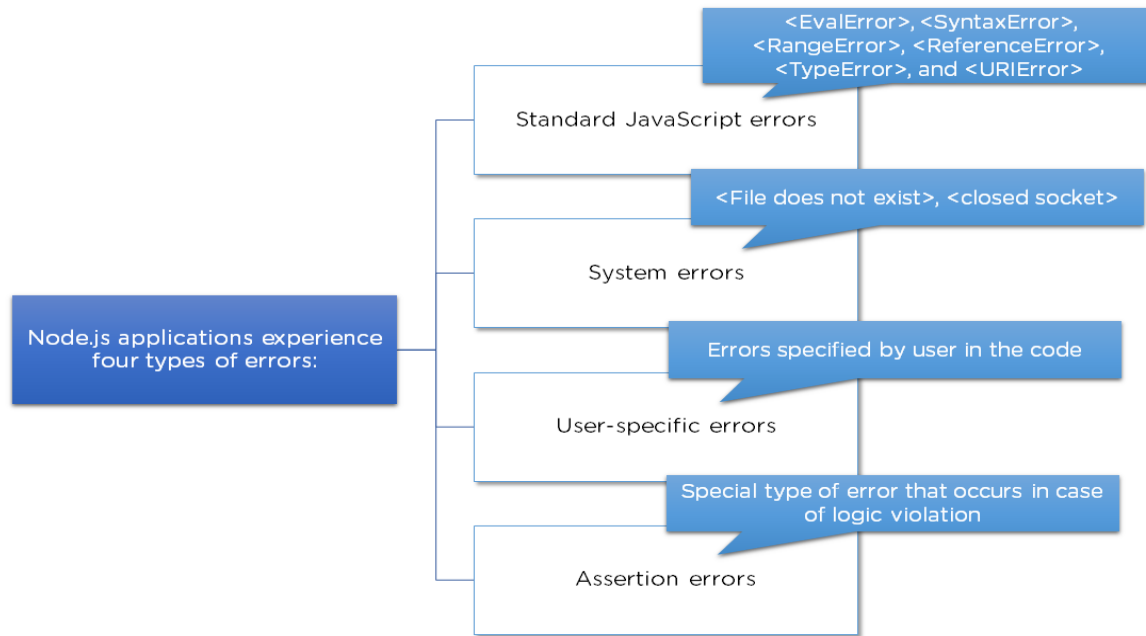


Fig: Node.js errors

Errors in Node.js are handled through exceptions. For example, let's handle the error that would occur when we divide a number by zero. This error would crash the Node.js application, so we should handle this error to continue with the normal execution of the application.

```
try {
  var m = 1;
  var n = 1/0;
}
catch (err) {
  // Handling the error here.
}
```

Fig: Node.js error handling

## Streaming

Streams are the objects that let you read data or write data continuously. There are four types of streams:

1. Readable: These are the types of streams from which data can be read
2. Writable: These are the types of streams to which data can be written
3. Duplex: These are both readable and writable streams
4. Transform: Streams that can manipulate the data while it is being read or written

## Buffer

Buffer is a module that allows the handling of streams that contain only binary data. An empty buffer of length '10' can be created by this method:

```
var buf = Buffer.alloc(10);
```

Fig: Node.js buffer

## Domain

The domain module intercepts errors that remain unhandled. Two methods are used for intercepting these errors:

1. Internal Binding: Error emitter executes its code inside the run method
2. External Binding: Error emitter is explicitly added to a domain via its add method

## DNS

DNS module is used to connect to a DNS server and perform name resolution by using the following method:

```
dns.resolve()
```

Fig: DNS resolve

DNS module is also used for performing name resolution without a network communication by using the following method:

```
dns.lookup()
```

Fig: DNS lookup

## Debugger

Node.js includes a debugging utility that can be accessed by a built-in debugging client. Node.js debugger is not feature-packed but supports the simple inspection of code. The debugger can be used in the terminal by using the 'inspect' keyword before the name of the JavaScript file. In order to inspect a file—myscript.js, for example—you can follow this method:

```
$ node inspect myscript.js
```

Fig: Node.js debugger



## Node.js Express Framework

Express is a flexible Node.js web application framework that provides a wide set of features to develop both web and mobile applications. It's a layer built on the top of the Node.js that helps manage a server and routes.

Now look at some of the core features of the Express framework:

- Used for designing single-page, multi-page, and hybrid web applications
- Allows developers to set up middlewares for responding to HTTP Requests
- Defines a routing table that is used to perform different actions based on the HTTP method and URL
- Allows dynamic rendering of HTML Pages based on passing arguments to templates

Now look at an example of a simple "Hello World" program developed using the Express framework to gain a better understanding of this framework.

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

app.get('/', function (req, res) {
  res.send('Hello World');
})

var server = app.listen(8081, function () {
  var host = server.address().address
  var port = server.address().port

  console.log("Example app listening at http://%s:%s", host, port)
})
```

- var express: Importing Express framework into our Node.js application
- app.get(): Callback function with parameters 'request' and 'response'
- The request object: It represents the HTTP request and has properties for the request query string, parameters, body, HTTP headers, etc.
- The response object: It represents the HTTP response that an Express app sends when it gets an HTTP request.
- The application will listen to the defined port, which in this case is "8081," and variables "host" and "port" will contain the address and the port, respectively.
- console.log: This is to show the address and port in the command prompt or terminal.

## Node.js Use Cases



Fig: Node.js use cases

## Netflix

### Netflix

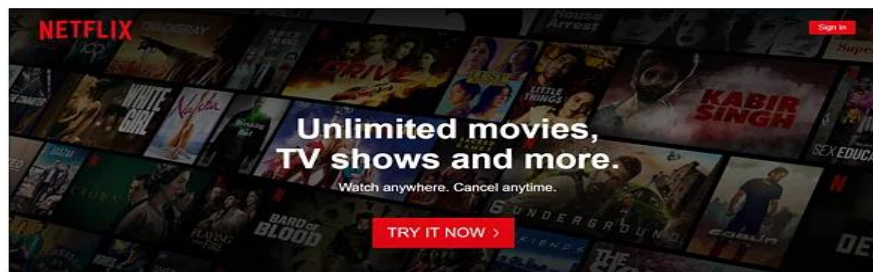


Fig: Netflix

Netflix, the world's leading online entertainment network with more than 167 million users, is one of many top companies trusting Node.js for their servers. The reasons why the company chose to use Node.js include:

- Application scalability
- Data-intensive application

## Walmart

### Walmart

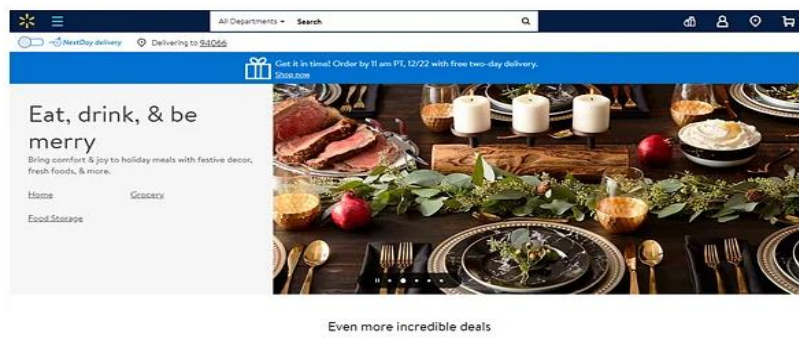


Fig: Walmart

Walmart is the world's largest company by revenue, with US\$ 559 billion in 2020, according to Forbes. Walmart chose Node.js because of the following attributes:

- Asynchronous I/O
- Efficient handling of concurrent requests

Uber

Uber

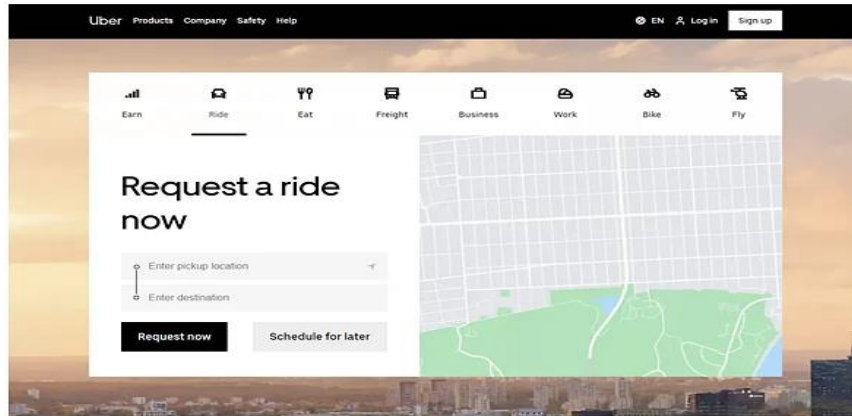


Fig: Uber

Uber is a U.S.-based, multinational ride-hailing company offering services that include peer-to-peer ridesharing, ride service hailing, and food delivery. The reasons why the company chose to use Node.js include:

- Asynchronous I/O
- Quick iterations
- Active open-source community

NASA

NASA

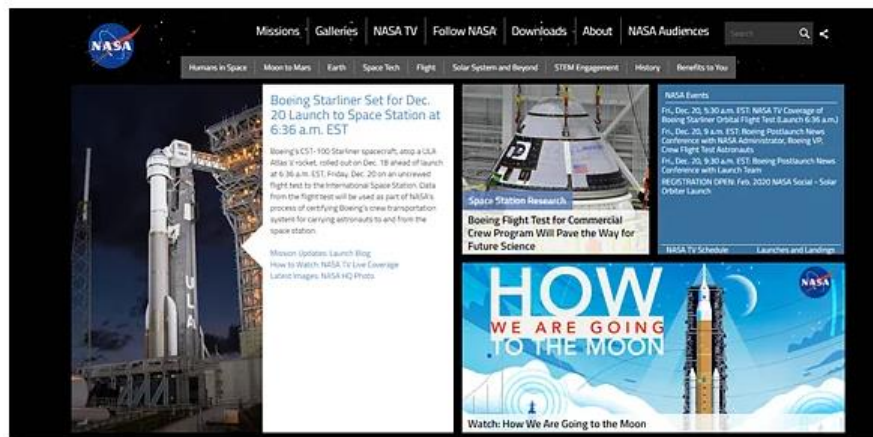


Fig: NASA

NASA, an independent agency of the United States Federal Government, is responsible for the civilian space program, as well as aerospace and aeronautics research. NASA chose to use Node.js for the following reasons:

- Reduced access times
- Ability to handle data-intensive tasks
- Capability to keep the server active 24/7

## Paypal

### Paypal



The safer, easier way to pay – in India and around the world.

Fig: Paypal

PayPal is a U.S.-based company operating a global online payment system that supports online money transfers, that is serving as an electronic alternative to traditional paper methods like checks and money orders.

PayPal chose to use Node.js for the following reasons:

- Extremely fast build times
- Fewer lines of code
- Ability to handle large amounts of data

## Medium

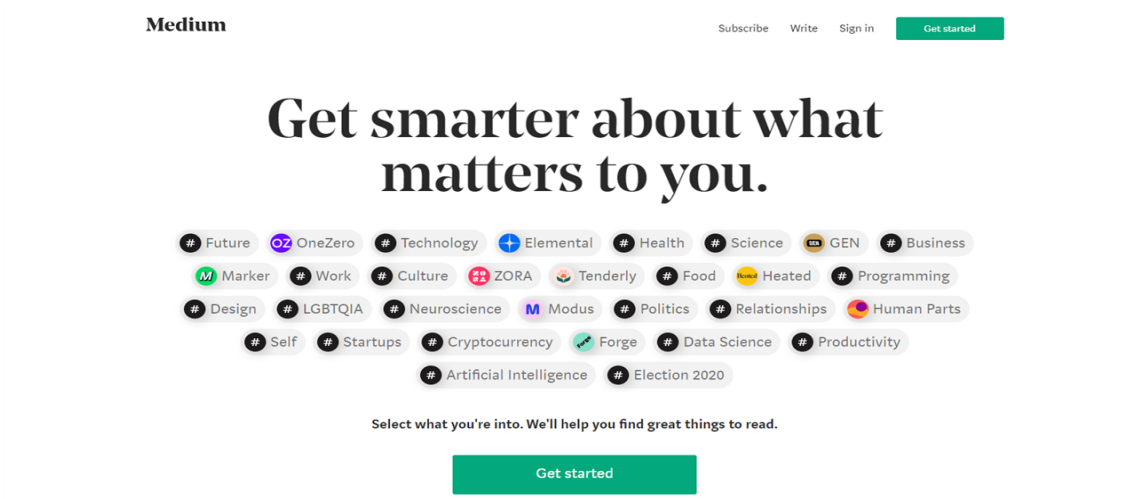


Fig: Medium

Medium is a popular online publishing platform developed by Evan Williams and launched in August 2012.

The reasons why the company chose to use Node.js include the following:

- Data-driven applications
- Ability to run A/B tests
- Simple server maintenance

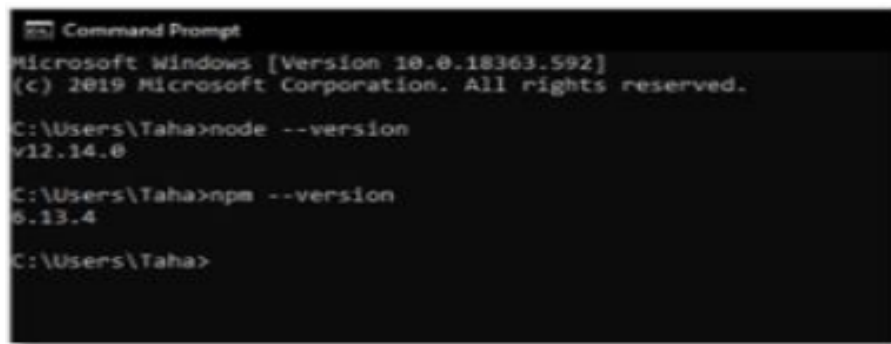
## NPM: Node Package Manager

NPM is a popular Node.js package library and the jewel in the crown of the Node.js community. It has millions of downloadable libraries, organized according to specific requirements, and is the largest software registry in the world. NPM is free. These libraries are growing fast to this very day, and they strengthen the Node.js community.

Node Package Manager provides two main functionalities:

- Online repositories for Node.js packages/modules, which are searchable on Node.js documents
- A command-line utility to install Node.js packages, do version management, and dependency management of Node.js packages

When you install Node.js, NPM is also installed. The following command in CMD can verify if NPM is properly installed: `npm --version`



```
Command Prompt
Microsoft Windows [Version 10.0.18363.592]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Taha>node --version
v12.14.0

C:\Users\Taha>npm --version
6.13.4

C:\Users\Taha>
```

Fig: NPM verification

Now that we have covered what is Node.js, Node.js Architecture, and NPM as part of this Node.js tutorial, let us now look at different Node.js Modules.

## What is Node Used For?

Here's a sample of ways that today's organizations use Node.js:

- Backend for social media networking
- Chat application
- Data streaming
- IoT application
- Single-page application

## Understanding the Popularity of Node.js

Node.js has attracted the attention of businesses and organizations from all sectors. This is hardly a surprise, considering its versatility and strong community support. As you can see from the earlier-mentioned use cases, there are some pretty big players that use Node.js, organizations and businesses like NASA, Uber, PayPal, and Netflix.

## Industry Trends

Node.js developers are in demand around the world due to the wide adoption of this JavaScript library. It is among the top 10 most in-demand jobs, according to Forbes.

- There were 98.9 million NodeSource Node.js Library downloads in 2020, according to Node Source.
- The use of Node.js in production has increased dramatically since its release in 2010.
- Node.js developers receive better salary options than other web technology developers. The average salary of a Node.js developer in India is ₹900,000 per year, and the average salary of a Node.js developer in the United States is \$115,000 per year!

With adopters such as Netflix, Paypal, and other tech companies, Node.js has seen an exponential increase in its use in web development.



Node.js also owes some of its popularity to its JavaScript association. Since JavaScript is the most popular language, as evident from the Stack Overflow survey in 2021, many developers can start working on the Node.js library without steep learning.



## Node.JS with Database

Node.js can be used with various databases to enable web applications to store data and perform operations with that data.

### MySQL

MySQL is among the more popular databases that can be connected to Node.js. It is the most popular open-source relational SQL database management system. It also is one of the best RDBMS used to develop various web-based software applications.

You can install the MySQL module using the following command prompt, and then add it to your file:

```
var mysql = require('mysql')
```

### MongoDB

MongoDB is also a popular database choice to connect with Node.js. It is an open-source document database and a leading NoSQL database. This database is often used for high-volume data storage.

You can install the MongoDB module using the following command prompt and then add it to your file:

```
var mongodb = require('mongodb')
```

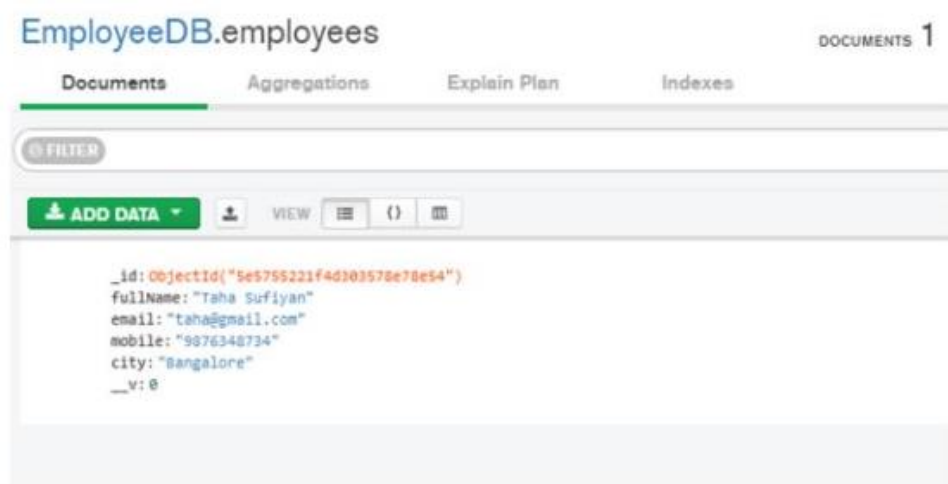


Fig: MongoDB Compass

MongoDB Compass is a tool that lets users connect to a database and view or edit the data from the dashboard.

This tool is installed while installing MongoDB.

The next section of this Node.js tutorial shows you how to create an application using Node.js.



## Create a Node.js application:

We are going to make our Node.js-powered weather application: Weatherly. This application will enable us to search for weather conditions anywhere in the world.

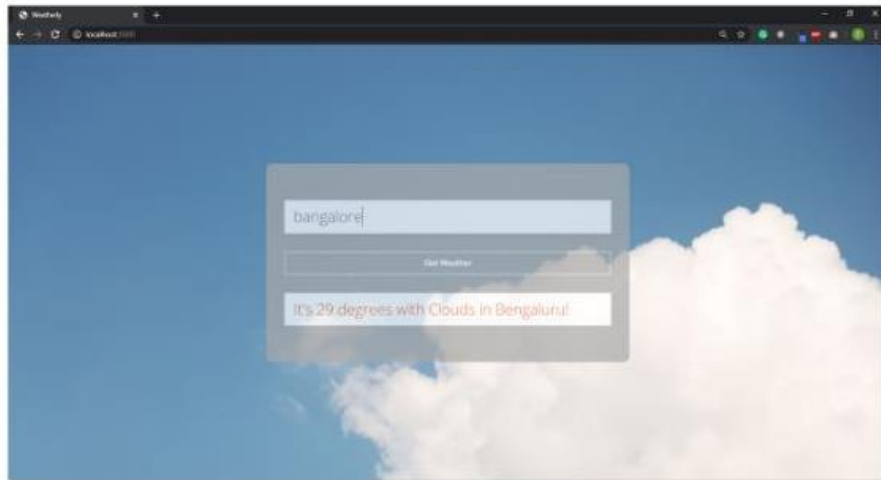


Fig: Weatherly Application

## Prerequisites

### Node.js Installation

1. Download the Node.js from <https://nodejs.org/en/download/>. Select the installer according to your operating system and environment.

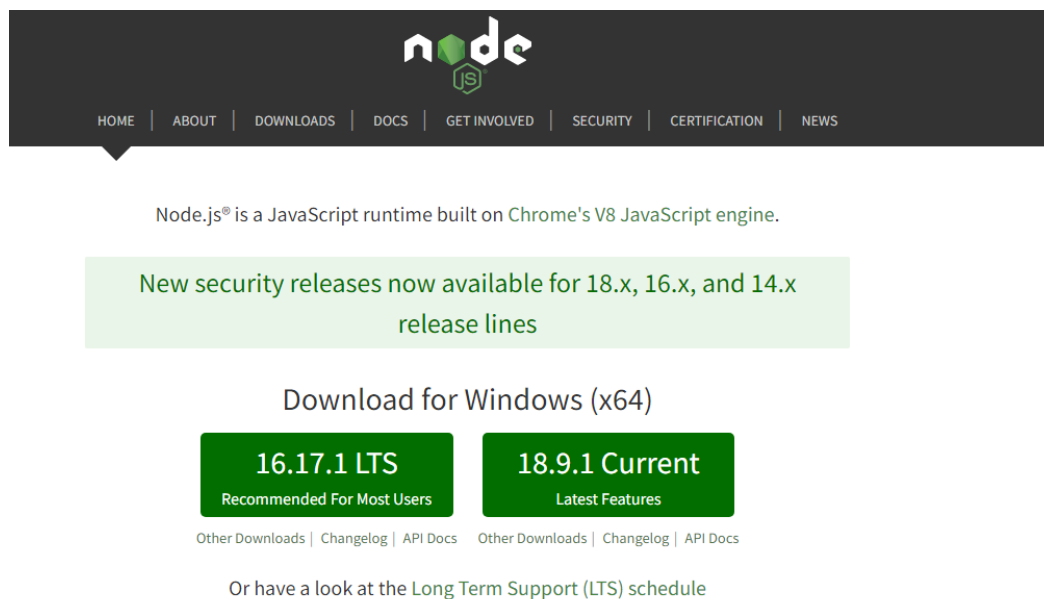


Fig: Node.js download page

2. Run the Node.js installer. Accept the license agreement. You can leave other settings as default. The installer will install Node.js and prompt you to click on the finish button.



Fig: Node.js setup

3. Verify that Node.js was properly installed by opening the command prompt and typing this command: `node --version`

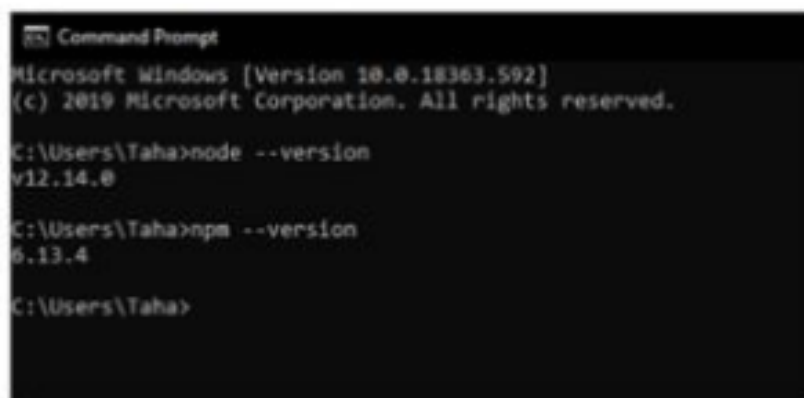


Fig: Node.js verification

4. When we install Node.js, NPM (Node Package Manager) is also installed. NPM includes many libraries that are used in web applications, such as React. Verify whether it is installed or not with the following command in CMD: `npm --version`

That's all we need to do to install Node.js in a Windows system successfully!

## API Setup

For this project, we'll be using the free OpenWeather API. Head over to [this link](#) and sign up for an account with an email and a password.

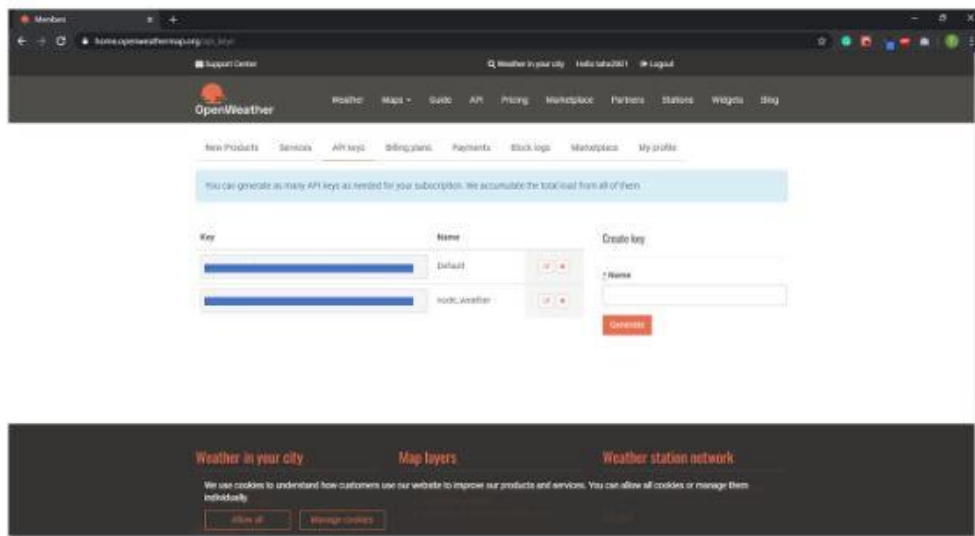


Fig: OpenWeatherMap dashboard

Once signed in, select the API Keys tab. Here, you can create a key on the right-hand side of the page. Enter a name for your application and select generate. The API key will appear on the left. We will use this key later in our code.

## Text Editor

Install a text editor of your choice. We are using [Visual Studio Code](#) in this tutorial, but you can also use other editors, like Atom and Sublime Text, if you are more comfortable with those.

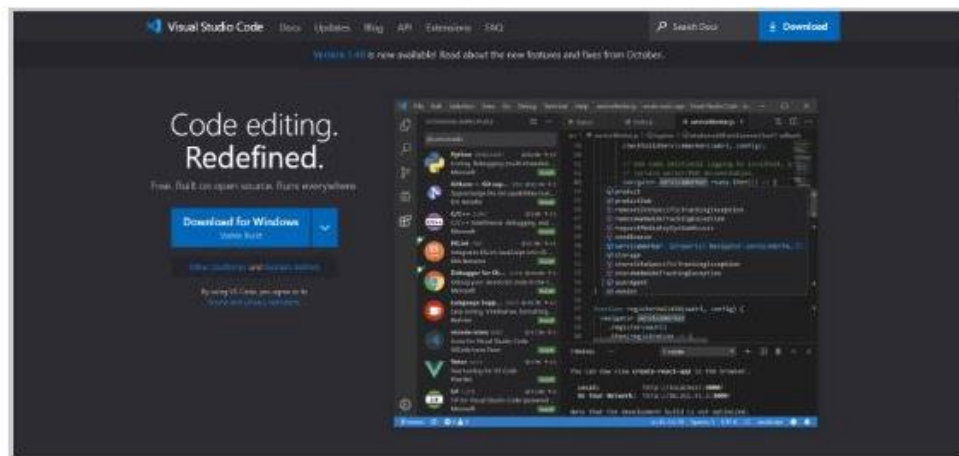


Fig: Visual Studio Code download page

## Project Setup

1. Create an empty directory named weatherly.
2. Open the newly created directory in VS Code, and inside the terminal, type `npm init` to initialize the project. Press the Enter key to leave the default settings as they are.

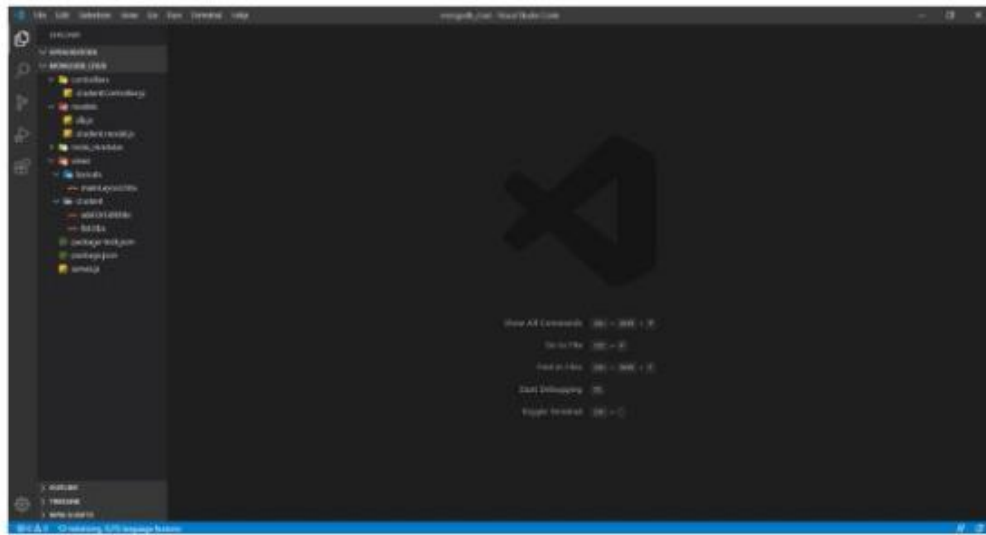


Fig: VS Code project

3. Within the weatherly directory, create a file named server.js, which will contain the code for our application.

### Server.js

Create a file called server.js in the project directory. This file in our application acts as the main server because it contacts OpenWeatherMap using the API key and returns the weather conditions of the inputted location.

Let's go ahead and learn what task each snippet of code carries out in this file.

1. `const express = require('express');`
2. `const bodyParser = require('body-parser');`
3. `const request = require('request');`
4. `const app = express()`
- 5.
6. `const apiKey = '*****';`
- 7.
8. `app.use(express.static('public'));`
9. `app.use(bodyParser.urlencoded({ extended: true }));`
10. `app.set('view engine', 'ejs')`
- 11.
12. `app.get('/', function (req, res) {`
13. `res.render('index', { weather: null, error: null});`
14. `})`
- 15.
16. `app.post('/', function (req, res) {`
17. `let city = req.body.city;`
18. `let url =`  
``http://api.openweathermap.org/data/2.5/weather?q=${input}&units=metric&appid=${apiKey}``

```

19.     console.log(req.body.city)
20.     request(url, function (err, response, body) {
21.         if(err){
22.             res.render('index', { weather: null, error: 'Error, please try again'});
23.         } else {
24.             let weather = JSON.parse(body)
25.             if(weather.main == undefined){
26.                 res.render('index', { weather: null, error: 'Error, please try again'});
27.             } else {
28.                 let weatherText = `It's ${weather.main.temp} degrees with ${weather.weather[0].main} in
                ${weather.name}`;
29.                 res.render('index', { weather: weatherText, error: null});
30.                 console.log("body:", body)
31.             }
32.         }
33.     });
34. })
35.
36. app.listen(3000, function () {
37.     console.log('Weatherly app listening on port 3000!')
38. })

```

- Line 1 - 3: We are importing three modules in the application: express, body-parser, and request. We can add these modules using the terminal inside the VS Code.



- Line 4: Create an Express object called app
- Line 6: Add the API key to our application
- Line 8: This line of code is used to access the CSS file we added in the public folder inside the project directory to make the application look better
- Line 9: Access the request body
- Line 10: Set up the EJS template engine with this line of code
- Line 12-14: Use the GET method for rendering the index.ejs (front-end HTML) file as a response
- Line 16 - 34: Use the POST method to request the server for weather conditions at a given location:
  - Store the city from the request body in the city variable
  - Send the URL with the city name and API key

- If there is an error, send an error message to be rendered on index.ejs
- Otherwise, send the weather conditions to be rendered on index.ejs
- Line 36 - 38: Set the application to listen at port: 3000, and log the message to the console

## Index.ejs

Instead of responding with text when someone visits the root page, we'd like to respond with an HTML file.

For this, we'll be using EJS (Embedded JavaScript). EJS is a templating language.

To add this feature to our application, we have to install it using the terminal the same way we added modules.

Use this command: `npm install --save ejs`

EJS is accessed by default in the views directory. Create a new folder named views in the application directory.

Within that views folder, add a file named index.ejs. This is how the project directory should look:

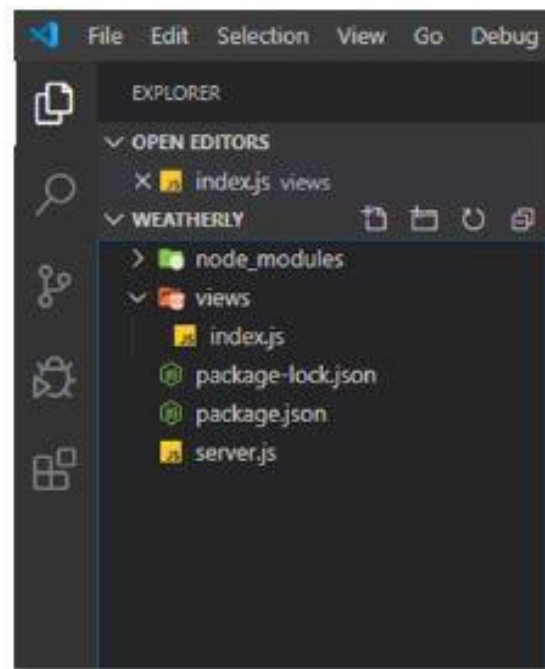


Fig: Project directory\_1

Now, let's go ahead and add the code for this file and then learn what exactly each line of code does.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Weatherly</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
          id="submit"
          type="submit"
          class="ghost-button"
          value="Get Weather"
        />
      </form>
      <% if(weather !== null){ %>
        <p><%= weather %></p>
      <% } %> <% if(error !== null){ %>
        <p><%= error %></p>
      <% } %>
    </fieldset>
  </div>
</body>
</html>

```

## Style.css

You can add the following CSS code to your application to make it look exactly like the one in this Node.js tutorial. You are free to tweak the CSS styles according to your preferences, too.



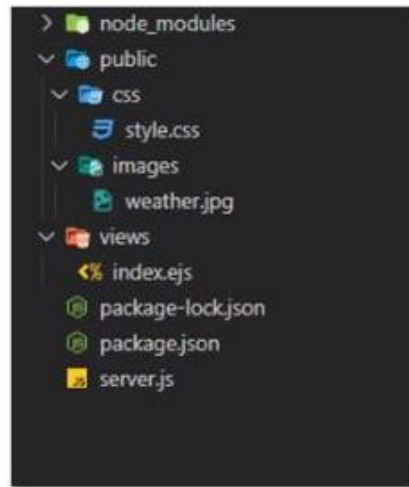


Fig: Project directory\_2

Add the following CSS styles in the style.css file and keep the file inside the public/css folder inside the project directory to make the styles work with the application.

```
body {  
    width: 800px;  
    margin: 0 auto;  
    font-family: 'Open Sans', sans-serif;  
    background: url("../images/weather.jpg");  
    background-size: cover;  
}  
  
.container {  
    width: 600px;  
    margin: 0 auto;  
    position: absolute;  
    top: 50%;  
    left: 50%;  
    transform: translate(-50%, -50%);  
    opacity: .7;  
    background-color: darkgray;  
    border-radius: 10px;  
}  
  
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;

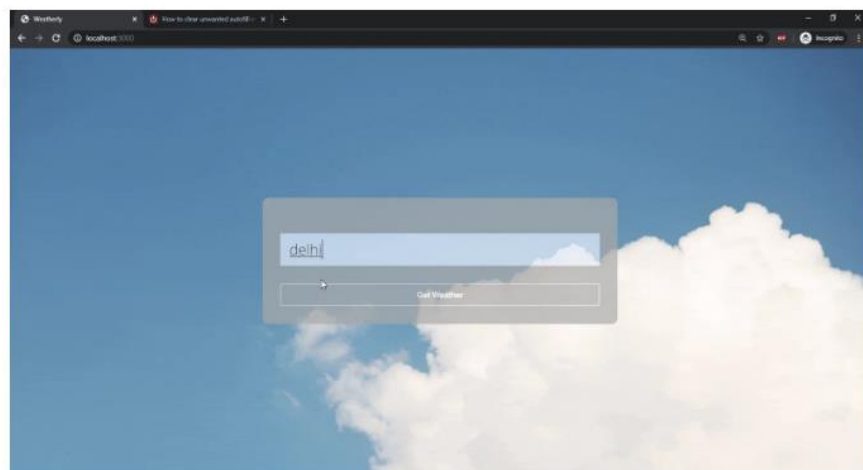
```

```

}
.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;
  color: white;
  font-weight: bold;
}
.ghost-button:hover {
  border: 2px solid #515151;
}
p {
  color: #E64A19;
}

```

## Final Result



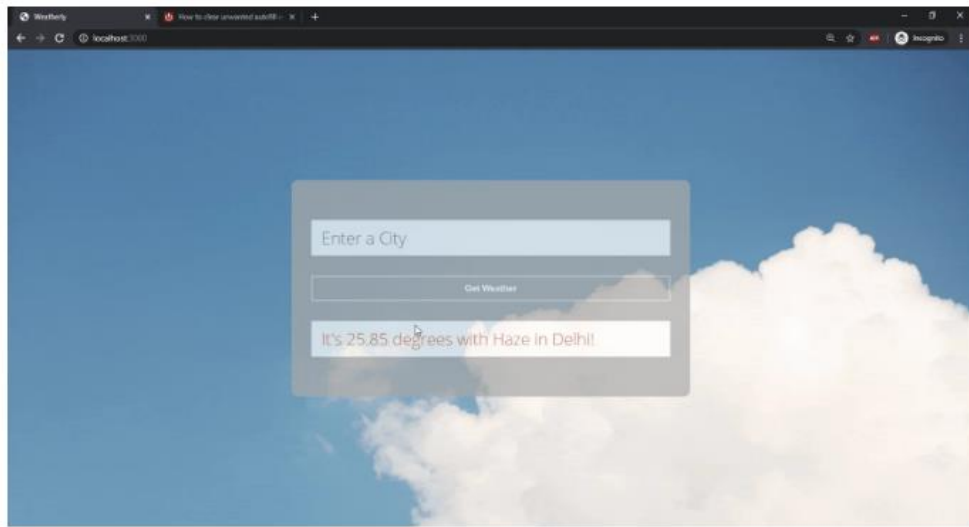


Fig: Weatherly Application

In the last section of this Node.js tutorial, let us look at some of the industry trends and demands associated with Node.js.

## Industry Trends

Node.js developers are in demand around the world due to the wide adoption of this JavaScript library. It is [among the top 10 most in-demand jobs](#), according to Forbes.

#### Fig: Node.js Source Report

- The total number of Node.js downloads [increased by 40 percent in 2018](#), according to Node Source
- The use of Node.js in production has significantly increased since its release in 2010

With adopters such as Netflix, PayPal, and other tech companies, Node.js has seen an exponential increase in web development.

The popularity of Node.js is also due to the fact that it's built in JavaScript. Since JavaScript is the most popular programming language, as evident from the survey conducted by [Stack Overflow in 2019](#), many developers can start working on the Node.js library without too steep of a learning curve.

Source: Stack Overflow Report 2019

- Node.js developers are being offered better salary options than other web technology developers

The average salary for a Node.js developer in India is ₹6,13,000 per year!

The average salary for a Node.js developer in the United States is \$104,964 per year!



To sum up all that's required in getting started with node.js, check out the Node.js tutorial video.