Intro – NodeJS

Till date, all the javascript files to be executed we need a .html file and we could run the script in a browser. Unlike Python, we cannot run the JavaScript program in a terminal. As the internet grew, we will have to have a server to do more stuff, more security, authentication, database etc. in order to do this, we had to learn Java, C++, Python, Ruby, C# .Net were used. To create a proper production level application, we need to learn another language. So, this was being a tedious job.

A guy name Ryan Dahl, using C++, C and JavaScript created a runtime and he named it **NodeJS**. If you have learned JavaScript, you can very easily transition into the server or the backend world. Using JavaScript and a familiar syntax we can create a server. We can use that server for various things. we can literally off load heavy lifting to the server. The main and important benefits is one language for front-end and back-end.

The moment NodeJS came-in ES6 also started developing. Earlier we were not able to access users file system. We cannot access it with browser based javascript. But we can now do it using NodeJS.

Difference in NodeJS.

It is **asynchronous**.

It is **Non-Blocking**. When a request comes in for a particular file, TomCat (Java Server) would search for the file and then once it is done then it would move to the next request. However, in NodeJS, it gives instructions to the server to find the file and it will move to the next request until the server finds the first requested file. Once it is done, it would read it and shared the content to the user.

Hello World = NodeJS

In JavaScript we would use console.log(‘Hello World’)

```
console.log('Hello World')
```

But in NodeJS, in terminal, we type

node script.js

We will see the output in the terminal.

We won’t have

- DOM manipulation.
- window object
- document as well
- Anything related to HTML and CSS

But all the concepts such as callback, closures, hoisting, anno… res, es6, classes, promises are still available as they are language specific features.

Create a package.JSON file.

What sought of information can we store in .json file?

It would have properties related to you project.

```
{
    "name": "Stock App",
    "version": "1.0.0",
    "author": "Yash"
}
```

Instead of typing all the details, we can enter **`npm init`** in the terminal and the terminal would keep taking details from us for the content of the details in the **`package.json`** file.

```
{
    "name": "sample-app",
    "version": "1.0.0",
    "description": "my awersome sample app",
    "main": "index.js",
    "scripts": {
        "test": "echo \"Error: no test specified\" && exit 1"
    },
    "author": "Yash",
    "license": "ISC"
}
```

`package.json` is a meta data file, that has all the information or data that we would use to build a project.

Generate Random Colors:

2 ways to do it:

1. You write your own logic
2. Save time by going through online repositories for Node packages. (3rd party packages) → https://www.npmjs.com/

In the npmjs.com we have multiple APIs for a random color generator. We have selected randomcolor. To get it, we need to write **`npm install randomcolor`** in the terminal.

```
const randomClr = require('randomcolor')

const myColor = randomClr()

console.log(myColor)
```

If we delete the third-party folder, we cannot manually re-install it. The best way is to use **`npm install`** command. The **`npm`** will go through the package.json file and all the packages that are listed in the dependencies will be re-installed.

Express:

Express JS is a web application framework work for NodeJS. For Python also Django and Flask exit for the same reason. Express is a fast, unopinointed, minimalist web framework for Node.

In the installing part we see the npm init options. To install express, **`npm i express`**. This will install a lot of different packages in the folder. This also adds an entry in the dependencies, **`"express": "^4.17.1"`**. First thing is to import random color as we would use it at some point. we need to import it and then call it and on it we can define certain part.

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

app.get('/', function(req, res) {
    console.log('Received Request')
})
```

The function will have two parameters.

req → has all the information related to the end user

res → response object, that we would use to send the data back

Now, after console.log we can add,

```
// defining route
app.get('/', function(req, res) {
    console.log('Received Request')
    res.send('Serverd Request')
})
```

We have defined a particular route. Now we have to start a server. To start our server,

```
app.listen(3000, function() {
    console.log('Server Started')
})
```

Now, we can test few things. To run the file, we would type **"node ."** in the terminal and the server would start.

If we enter localhost:3000 in the address bar of the browser, we would see Served Request. Since randomClr is imported, we can type,

```
app.get('/', function(req, res) {
    console.log('Received Request')

    const color = randomClr()
    res.send(color)
})
```

This will give us an output of #color in the web page. In the network time, we can see the request url was http://localhost:3000/ and it was a get request. Finally in the response we are receiving a string, which is coming from the randomClr() package.

We are getting the response on the home route, so if we change the source code from '/' to '/randomColor' and then change the route path in the address bar of the browser as well to http://localhost:3000/randomColor it will give us the same output.

```
// defining route

app.get('/randomColor', function(req, res) {
    console.log('Received Request')

    const color = randomClr()
    res.send(color)
})

app.listen(3000, function() {
    console.log('Server Started')
})
```

In your system, you have your browser. Using NodeJS you have created a server program in your system as a process. If we run notepad in the system it would be another process. We have created a NodeJS server in our system currently and the chrome browser is able to communicate with that server using http protocol.

In real world, after the application is developed, we upload the entire application into cloud or another system and it would be running from there. When we do this, then we will get a proper url.

Local host is a special IP address. Local host is 127.0.0.1. This is a special ip address which tell browser not to create a request anything in the internet, rather create a request for the server running in our system.