Getting Started with Node.js

Estimated Time: 15 minutes

Hello there. You came here because you want to learn about Node.js, a very popular technology for building web apps. Traditionally, JavaScript was only able to run on a browser, but Node.js allows JavaScript to run on a server. You can call Node.js a runtime, a platform, or an environment that allowed JavaScript to run on a server. Which allowed apps to be entirely written in JavaScript.

> Node.js is **not** a framework.

Anyhow, I welcome you to this Guided Project.

What you will achieve

1. Use Node.js and NPM to create an application that prints "Hello World".
2. Run the application through the Skills Network extension.
3. Add some style to your application.

What You Will Need

- A computer, obviously
- Willingness to learn and challenge yourself!

Additional Notes

- **This app is yours.** Feel free to add more pages, and features after this project to make this project more awesome.
- **Google is your best friend.** You might face bugs and some things may not work on your machine even if you follow everything step by step. This is where you willingness to learn comes in, google the error you are seeing and try to figure it out.
- **Have FUN**

The Set Up

1. `Node.js` and `NPM` should already be installed in your Cloud IDE. We already know what Node.js is, but what is `NPM`? `NPM` is your package manager, it will handle all your packages and frameworks that you need in your project. To import a package, you can run `npm install <package name>`.

2. Let's run some command to get started. First, run the following command in the terminal:

```
1
```

```
npm init
```

Copied! Executed!

This command will initialize your project, prompting you to input some metadata about your project such as the name of the project, description, author, etc. To leave it as default or empty, just press ENTER. The most important field is the `entry point` which has to match your root JavaScript file. Keep it as default, `index.js`. We will be creating this file later.

> **NOTE:** Once you've finished, `package.json` should be created.

Every Node Package has package.json. It holds all the metedata of the project. It's like the ID of the project. For more: https://nodejs.org/en/knowledge/getting-started/npm/what-is-the-file-package-json/

```
theia@theia-lavanyas:/home/project$ 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 init` 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: (project) awesomeapp
version: (1.0.0)
description: My first Node js app
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (MIT)
About to write to /home/project/package.json:

{
  "name": "awesomeapp",
  "version": "1.0.0",
  "description": "My first Node js app",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "MIT"
}


Is this OK? (yes) yes
```

3. Let's install Express using npm. Run:

```
1
```

```
npm install express
```

Copied! Executed!

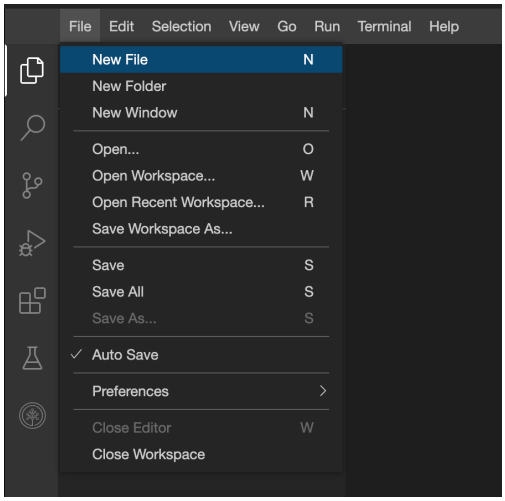
Express is a good example of a package that wouldn't be possible without Node.js allowing JavaScript to run on a server. Express provides an easy way to set up integral parts of webapps like routing and handling HTTP requests.

This should have created `package-lock.json` and `node_modules` directory. Now, we can now use Express in our application!

> All packages installed by npm will be inside `node_modules` directory. You will be able to find a `express` directory inside `node_modules` which has been just installed. You can see how each package is its own project and we are literally importing the code to be re-used for our project.

Let's write some JavaScript!

1. In Cloud IDE, select `File > New File` as seen below.



2. Choose a name for the file. In our case, we will name it `index.js` to match the "Entry Point" in our package.json.

3. In the file, copy and paste the code below. This code will create a web app running on our localhost and will say "Hello World".

```
1
2
3
4
5
6
7
8
9
10
11
```

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Your app is live at ${port}`)
})
```

Copied!

4. Let's understand what we've written.

On the first line, we are importing the framework into our code that we've previously installed using npm. Express makes it really easy to start the server. On line 5, we are declaring how our server will respond to `/` route. Here we are just returning text 'Hello World!'.

5. Let's run the code! Inside Node.js, you can run a JavaScript file by running `node <file name>`. In the terminal run the following to run `index.js`:

```
1
```

```
node index.js
```

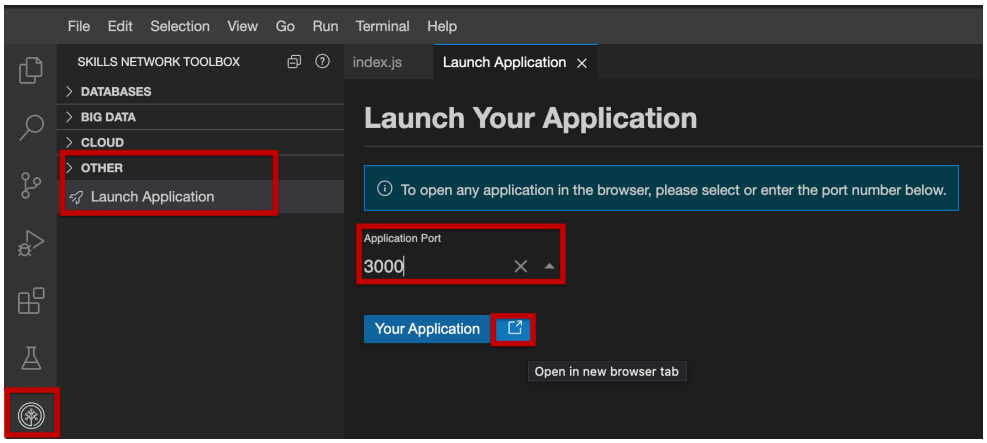
Copied! Executed!

Your output should be similar to the following:

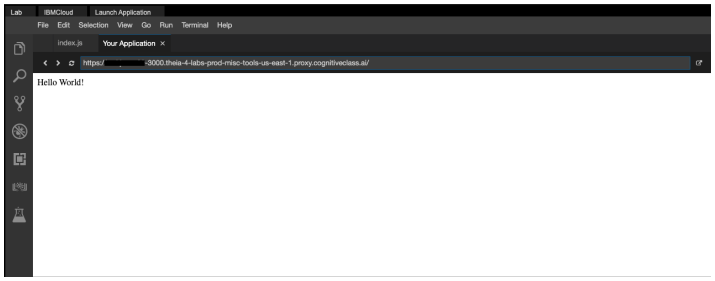
```
theia@theia-lavanyas:/home/project$ node index.js
Your app is live at 3000
```

Click on the `Launch My Application` button below or navigate to the Skills Network Extension and select "OTHER". Click "Launch Application". Under "Application Port", type in the port number. In this case, it is 3000. Press "Your Application".

Launch My Application



Your app should be able to see your app!



6. When you are done, go to your terminal and `CTRL-C` to stop the app.

Add Some Style!

Currently, our app is too plain. Let's add some style to this page.

We are going to use

1. Create a new folder (`File > New Folder`) and name it `views`. Inside, create a new file and name it `helloworld.html`. It's good practice to create folders for different purposes. It is general convention to name the folder `views` and store all html files.

2. Let's copy the following HTML code to `helloworld.html`.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

```
<!DOCTYPE html>
<head>
    <title>HTML and CSS "Hello World"</title>
    <style>
        body {
            background-color: #2dc0ff;
        }

        h1 {
            color: black;
            font-size: 30px;
            font-family: Menlo, Monaco, fixed-width;
        }

        p {
            color: rgb(27, 124, 184);
            font-family: "Source Code Pro", Menlo, Monaco, fixed-width;
        }
    </style>
</head>
<body>
    <h1>Hello World!</h1>
    <p>This is a very basic "Hello World" example made up in HTML and CSS. See if you can change the size of the header text above.</p>
</body>
</html>
```

Copied!

This is a typical HTML code block. HTML are built of *tags*. Tags start with `<>` and `</>`. For example, header tag goes `<h1>` and ends with `<h1>`. Let's go through each tag one by one.

Every HTML file starts with `<!DOCTYPE html>`, not sure why let's move on to the `<head>` tag. The `<head>` is where all your metadata goes, like the title, and the style. This title will be shown on the browser's title bar or the tab bar. The style allows you style certain tags. For example, we see that we set the `<body>`'s background colour to be #2dc0ff.

After the `<head>` tag, there is a `<body>` tag and this is where you define the page itself. You will see various tags like `<h1>`, `<p>`, `<li>` and these each have there own purpose to them.

3. Now let's change `index.js` such that it will render the HTML.

a. We are going to import `path` library. Add the following to the top of the code. `Path` makes it really easy to interact and access different files in the project.

```
1
```

```
const path = require('path');
```

Copied!

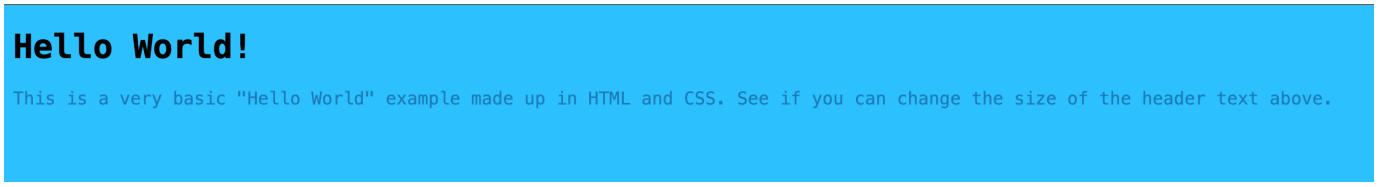
b. Change line 6 to render the file instead.

```
1
2
3
4
```

```
// Before
res.send('Hello World!')
// After
res.sendFile(path.join(__dirname, 'views/helloworld.html'));
```

Copied!

c. Start the app again and see the changes. It should look something similar to this:



d. Now, add your own style and features to this web app! (Optional)

Conclusion

Great job. You did it! You officially know how to build a Node.js app. We learned what Node.js is and how to use NPM to import useful packages and libraries. We ran our app using Express and finally, we also added some style to the app by rendering an html file instead of plain text.

I hope you enjoyed as much as I did creating this project.