

# JavaScript Frameworks

## Section 1: Node and Express

---

### 1.1 Introduction to Node and Express

Node.js is a platform built on Google's V8 JavaScript runtime engine and allows to build fast and scalable network applications. This allows developers to use the JavaScript language on the backend.

Node.js can be installed from their website (<https://nodejs.org/en/>) and this also installs the Node Package Manager (NPM) which allows us to download packages/modules/libraries for our projects.

Express is a npm module/library that provides a minimalist web framework for Node which provides a robust set of features for web and mobile applications. To install express we use the following command to run in the terminal within our project directory.

```
$ npm install express --save
```

This will save express as a package dependencies within our package.json file as well as the associated files in the node\_modules directory. The --save flag saves the package as a dependency of our project while the --save-dev saves packages as a development dependency.

To check whether the following Node, NPM and Express are installed on your machine we would run the following commands to display the versions installed.

```
$ npm --version $ node --version $ express --version
```

The Node REPL environment allows us to write JavaScript in the command line i.e. terminal. REPL stands for Read Evaluate Print Loop. We can write simple arithmetic and functions in the terminal and have the output returned back to us. To escape the Node Shell by pressing control + c at the same time on our keyboard twice.

Node is also able to serve files. If we have a .js file we can use the command node followed by the file path/name and the result of the file will be printed to the terminal window. Below is an example of a very simple Hello World application which is a proof of concept.

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

app.get('/hello', function(req, res) {
  res.send('hello from index.js');
});

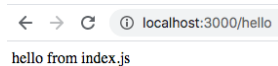
app.listen(3000);
```

The require function is used to import/access the express module. We use this to assign the express variable the entirety of the express module. We then assign the app variable to an instantiation of express. Anything we chain onto the app variable is a method from the express library.

The `.get()` method takes two parameters the first is the end route and the second is a callback function. The callback function takes in two parameters which is the request and the response. When we hit the `/hello` endpoint in the browser it will fire off this anonymous function. The function will attach to the `res` object the `.send()` method which will send back a string that says `'hello from index.js'` to the browsers terminal.

The `.listen()` method takes in one parameter which is the port number to listen on. Once the file is created we can run the following command in the terminal:

```
$ node index.js
```



A screenshot of a web browser window. The address bar shows 'localhost:3000/hello'. The page content displays 'hello from index.js'.

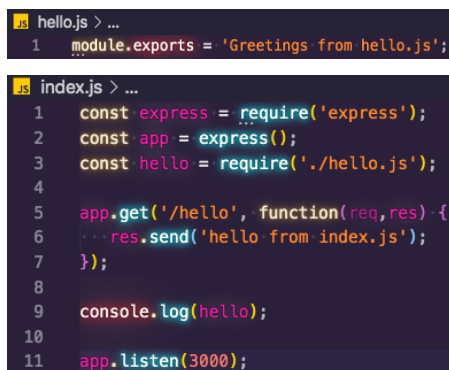
This will run the code inside of the `index.js` file and when it does we will see a blinking cursor in the terminal. This means the web server is running on `localhost:3000`. If we navigate to <http://localhost:3000/hello> in the browser we should see `'hello from index.js'` printed in the browser. This indicates the application route was a success.

---

## 1.2 Nodemon

Our code will be much easier to manage if we are able to separate the code into different files. Ideally we would separate our code into files that contain no more than one function or component. To handle this modularity JavaScript comes equipped with imports and exports.

In order to use function from File B in File A we have to export the function from File B and import the function into File A. Below is an example of exporting a string from `hello.js` to `index.js` using node's `module.exports` and `require` keywords.



A screenshot showing two code files. The first file, `hello.js`, contains a single line: `module.exports = 'Greetings from hello.js';`. The second file, `index.js`, contains the following code: `const express = require('express');`, `const app = express();`, `const hello = require('./hello.js');`, `app.get('/hello', function(req, res) { res.send('hello from index.js'); });`, `console.log(hello);`, and `app.listen(3000);`.

The `module.exports` puts a wrapper around the string in a nice little package which makes it so that we can transfer it from one file to another. On line 3 of the `index.js` file we are declaring a variable called `hello` and then assigning its value to everything that is inside of the `hello.js` file using the `require` function.

The `hello` variable is used to print the string to the terminal window/console. This demonstrates the successful export/import from one file to another.

In addition to this we would want to install a package called `nodemon`. `Nodemon` is a package that updates the browsers without restarting the server. Therefore, as we work on developing our files, instead of having to restart our server every time we make a change to see the changes, `nodemon` will do this for us automatically without having to do any extra work.

To install `nodemon` we would want to go to the terminal window and run the following command while `cd` into the root project directory. This will save `nodemon` as a development dependency module/package in our `package.json` file.

```
$ npm install --save-dev nodemon
```