SIT737 Prac 1

# Know your tools.

When writing our applications, there is a time when we need to focus on performance (c++), other times when we require higher interoperability with certain technologies (python and machine learning), other times, we are looking for the jack of all trades.

For this reason, in this unit, we will learn about NodeJS.

Definition from the Node foundation

“Node.js is a JavaScript runtime built on [Chrome's V8 JavaScript engine](https://developers.google.com/v8/). Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, [npm](https://www.npmjs.com/), is the largest ecosystem of open source libraries in the world.”

We have decided to use Node in order to streamline our application development, it is a very good programming language which is very easy to use and well supported by the community.

You can easily find many tutorials online, I suggest to find one which is more incline to your style, and explore it as much as possible. There also many libraries available, using NPM, which is node’s package manager.

Creating complex applications is quite simple, scalability is well handled and there is high support from cloud vendors as well



Tools

**IDE** – suggested Webstorm , Visual Studio Code , or anything you are prophicent with. Lab computers will come preloaded with VS Code and Webstorm.

**Node JS** – make sure Node JS is installed in your machine, all labs’ machines should be preloaded. If you are using your personal machine, head to <https://nodejs.org> and download the latest one.

**Git** – Git is our repository manager tool. Labs machines are preloaded, if you are using your own, make sure to install from <https://git-scm.com/> Also create a github account if you don’t have one.

**Terminal** – This is really your bread and butter, while the most ancient and perhaps archaic tool, the terminal is really where everything happens. I recommend to use Powershell inside windows, or Iterm on MacOs. These both use linux syntax. Get used to the most common commands. Here is a cheat sheet <https://learncodethehardway.org/unix/bash_cheat_sheet.pdf> . At least, learn the basic, such as **ls** and **cd** to move around.

# Node JS intro

# Setting up

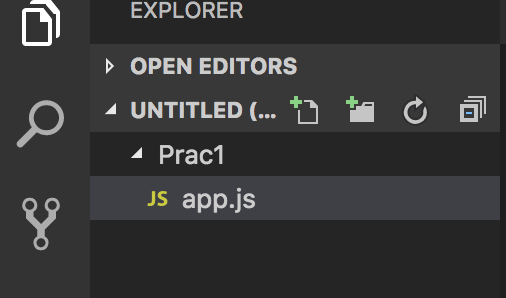
Let’s start by creating a folder where we are going to be working from. Do this in your C drive. Normally we would do it from the H drive, but this being a network drive, it may become very slow, therefore, work from your local machine.

Create a folder called **SIT737-yourStudentNumber** and inside this folder, create a new folder called **Week1**.

Open your IDE – visual studio code. Go to file and choose **Open Workspace**  or **Open Folder** (will change depending on your version).

Right click your workspace in the list of folders

Choose **New File** and call it **app.js** ,this will identify the file as a java script file.

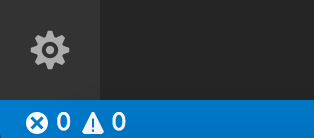


In our newly generated file, let’s take a trip down memory lane and start with the usual

console.log("Hello World")

This is the classic console output used in javascript. If you are viewing a web page, this would appear in your browser console, if running on your terminal, then it will output to the terminal console.

**Save the file** and try to run it. In order to do so, open your terminal, you can do so from within the IDE, by clicking on the small X button at the bottom left, and then choosing terminal among the tabs, or simply run the powershell and navigate into your directory.

This will bring up a number of tabs, choose the terminal tab.

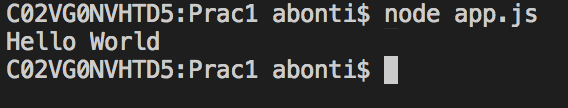
You can now run the application, before you do so, just verify that you are in the right place, type **ls** or **dir** if you are using the highly not recommended windows console.

If everything goes according to plans, you should be able to see your file listed.

Now run the command

**node app.js**

This will execute the node application (app.js is an application) and you should see the outcome on your console.



Congratulations on running perhaps, your first node application.

# Functions

Everything in javascript can be implemented as an object, therefore, a function is a function as well. A function is a procedure call stored as an object, that can be called from anywhere in the application (within scope visibility).

Important things to remember :

* A function has a name
* A function takes 0 or more arguments
* A function can return a value – does not implicitly return one
* JS is non blocking, calling a function will not block the execution.

Let’s create a simple function which outputs a message in some sort of “admin friendly” format.

var log=function(msg){

console.log("[Log] : ",msg)

}

log("hello world")

log("welcome to SIT737")

Save your application and run it again. Notice the difference?

Of course, this is not very exciting, but we need to make sure we all know our basics.

We are now going to see a bit more action.

var adder=function(first,second){

var sum=first+second

return sum

}

var log=function(msg){

console.log("[Log] : ",msg)

}

log("hello world")

log("welcome to SIT737")

log("The sum is "+adder(5,6))

A function calling another function, and the use of the **return** statement. Using a return statement in this way, allows us to sync our functions, otherwise it would return undefined.

One last thing, which is extremely important when working in a ASYNC environment, is to understand that NodeJS is non blocking.

Finally, try this code

var async = function(){

setTimeout(function(){log("I am coming out later although I have been called before the next one")},2000)

}

var adder=function(first,second){

var sum=first+second

}

var log=function(msg){

console.log("[Log] : ",msg)

}

log("The sum is "+adder(5,6))

async();

log("This is going to come out before the previous one")

The async function simulates a request made to a webservice, something is called , and we receive that information at a later stage. This is solved with promises and more advanced paradigms, which we will explore at a later stage.

# An Express Webserver

Node is a great language because it allows us to bring functionality. In creating our first webserver, we will also explore the use of packages.

To work with advanced structures, we relay on libraries, this time for example, we are going to be using express, a very popular library to create web servers using nodejs.

Before doing that, create package.json file. This is just a normal file, you can create it the same way we created the previous file, and populate it with the following content.

{

"name": "node-server",

"description": "A sample Node.js app using Express 4",

"dependencies": {

"express": "\*"

},

"author": "Alessio Bonti",

"license": "MIT"

}

This is basically a description of our app, which also defines the packages needed to run it. In this case, we only have one dependency, **express** .

This file only describes the application and provides information on how to build it.

In order to make use of it, we need to “build” the application. We do so by running the **npm install** command.

In your terminal, run the command, it should start downloading various files, and creating a folder which will contain such files (it may be hidden).

Create a new javascript file and call it **server.js** , and use the following code

var express = require('express')

var app = express()

// respond with "hello world" when a GET request is made to the homepage

app.get('/', function (req, res) {

res.send('hello world')

})

// list to a particular port

app.listen(3000)

Run the application and open your browser, point at localhost:3000 . Congratulations, you have now written your first web server in node!

**Exercise**

This was a very basic exercise, now create your own function to provide more interactivity.

What I want to see :

A logging functionality – just on screen – create a function that logs the main activities with the time as well.

For example

11:22:21 [Server Activity] : Web server started

11:22:30 [Server Activity] : Page has been requested

To do this, work with another person, and discuss what should be the best way of solving this simple problem.

Also, start including some comments in your code.

Use google to help you find the answer, stackoverflow is one of the most used resources by software engineers everywhere, great software engineers are those who answer those questions.

We call this, pair programming, work with another person and learn from each other. Use one computer only if it’s simpler.

Next head to the following

<https://www.ibm.com/watson/developer/> and look at the available Products and Services.

Setup a group of three quickly, and explore the offering.

Start thinking of how you could use these to create an application, do not worry about complexity!

Your group will be presenting it next week!