The Complete Developers Guide

Section 1

INSTALLING & EXPLORING NODE.JS

Installing Node.js

We can visit https://nodejs.org/en/ homepage to download node.js for our operating system. There are two versions:

The LTS version (Long Term Support) and the Current release version.

It is advisable to download the Current release, as this will contain all the latest features of Node.js. As at August 2019, the latest version is 12.7.0 Current.

We can test by running a simple terminal command on our machine to see if node.js was actually installed correctly:

```
:~$ node -v
```

This will return a version of node that is running on your machine and will indicate that node was installed.

If we receive command not found, then this means node was not installed and we need to reinstall it.

Installing A Code Editor

A text editor is required so that we can start writing out our node.js scripts. There are a couple of open source code editors (*i.e. free*) that we can use such as Atom, Sublime, notepad++ and Visual Studio Code to name a few.

It is highly recommended to download and install Visual Studio Code as your code editor of choice as it contains many great features, themes and extensions to really customise the editor to fit your needs. We can also debug our node.js scripts inside of the editor which is also a great feature.

Links:

https://code.visualstudio.com/

https://atom.io

https://www.sublimetext.com/

What is Node.js?

Node.js came about when the original developers took JavaScript (*something that ran only on browsers*) and they allowed it to run as a stand alone process on our machine. This means that we can use the JavaScript programming language outside of the browsers and build things such as a web server which can access the file system and connect to databases. Therefore, JavaScript developers could now use

JavaScript as a server side language to create web servers, command line interfaces (CLI), application backends and more. Therefore, at a high level, node.js is a way to run JavaScript code on the server as opposed to being forced to run on only the client.

> *"Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.*"

There are all sorts of JavaScript engines out there running all major browsers, however, the Chrome V8 engine is the same engine that runs the Google Chrome browser. The V8 engine is a open source project. The job of the engine is to take JavaScript code and compile it down into machine code that the machine can actually execute. The V8 engine is written in the C++ language.

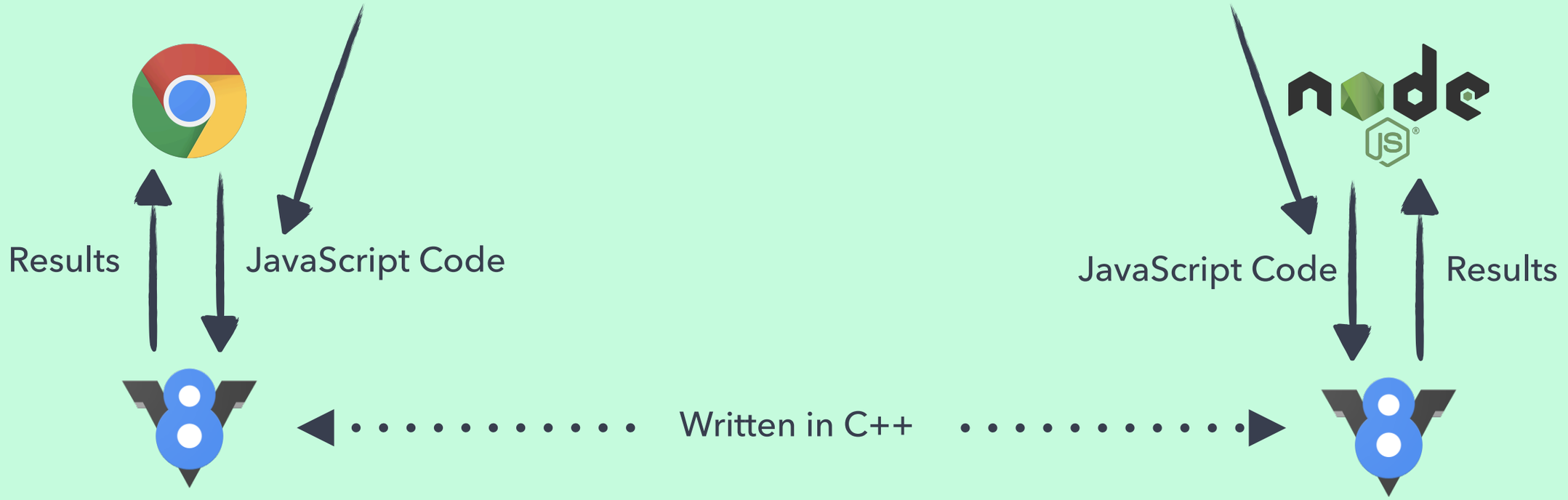
Node.js is not a programming language. The runtime is something that provides custom functionality i.e. various tools and libraries specific to an environment. So in the case of Chrome, the runtime provides V8 with various objects and functions that allow JavaScript developers in the chrome browser to do things like add a button click events or manipulate the DOM. The node runtime provides various tools that node developers need such as libraries for setting up web-servers integrating with the file system.

The JavaScript is provided to the V8 engine as part of the runtime. JavaScript does not know the methods e.g. read file from disk or get item from local storage but C++ does know the methods. So we have a series of methods that can be used in our JavaScript code which are, in reality, just running C++ code behind the scenes.

Therefore, we have C++ bindings to V8 which allows JavaScript, in essence, to do anything that C++ can do. Below is a visualisation to demonstrate exactly what is happening with the V8 JavaScript Engine:

JavaScript (Chrome)	C++
localStorage.getItem	Some C++ function
document.querySelector	Some C++ function

JavaScript (Node.js)	C++
fs.readFile	Some C++ function
os.platform	Some C++ function



We can run the following code in our terminal

```
:~$ node
```

What we get by running this command is a little place where we can run individual node JavaScript statements which is also known as repl (*read eval print loop*). These are not bash commands. All of the core JavaScript features we are use to are still available when using node.js because those are provided by the V8 engine itself.

It is important to note that some of the JavaScript features available in the browser are not available in node. For example, Chrome has a object called window which provides all the different methods and properties we have access to. This makes sense in the context of JavaScript in the browser because we actually have a window to work with. This will not work on node because window is something specifically provided by the Chrome runtime when JavaScript is running in the application. Node does not have a window and it does not need window and therefore window is not provided.

With node we have something similar to window; we have a variable called global:

```
:~$ node
[> global
```

Global stores a lot of the global things we can access such as the methods and properties available to us. The browser does not have access to the global variable.

Another difference between the browser runtime and the node runtime, is that the browser has access to something called a document. The document allows us to work with the DOM (document object manager) which allows us to manipulate the document objects on the page. Again, this makes sense for a browser where we have a DOM and it does not make sense for node where we do not have a DOM. In node we have something kind of similar to document called process. This provides various methods and properties for the node process that is running (e.g. exit() allows us to exit the current node process).

```
[> process.exit( )
```