Construction of User Interfaces (SE/ComS 319)

Ali Jannesari

Jinu Susan Kabala

Department of Computer Science

Iowa State University, Spring 2021

INTRODUCTION TO NODE.JS

Node.js

- Open-source, cross-platform **JavaScript** run-time environment that executes JavaScript code **server-side**
  - Historically JavaScript used for client-side programing
- "JavaScript everywhere" paradigm (popular)
  - Unifying web application development
  - Same language for server-side and client-side scripts.

A JavaScript runtime environment running Google Chrome's V8 engine

Goal is to provide an easy way to build scalable network programs

Why Node.js?

- Non-blocking I/O (asynchronous calls)
- V8 Javascript Engine
  - V8 is Google's open-source high-performance JavaScript engine, written in C++ and used in Node.js
- Single Thread with Event Loop
- 40,025 modules: JavaScript libraries you can include in your project
- Different platforms: Windows, Linux, Mac,…
- 1 Language for Frontend and Backend
  - Core in C++ on top of V8
  - Rest of it in Javascript
- Active community

Node.js – Libraries

standard lib

process.argv // returns command line arguments
console.log
setInterval(callback, time) // calls a function at specified intervals
require(library)

fs

Readdir // reads the contents of a directory
readFile // read entire file
readFileSync // read entire file (blocking)
createReadStream //read in chunks

path

Extname // get the extension from a file path

Node.js – Example 'Hello World!'

```
var http = require('http');

//create a server object:
http.createServer(function (req, res) {
  res.write('Hello World!'); //write a response to the client
  res.end(); //end the response
}).listen(8080); //the server object listens on port 8080
```

include a module (library), use the **require**() function with the name of the module

Use the **createServer**() method to create an HTTP server

Represents the request/response from/to the client

Writes "Hello World!" if a web browser tries to access your computer on port 8080

Node.js – Example 'Hello World!' (2)

- Create a file named "**app.js**"

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

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

- Run your web server using **node app.js**
- Visit http://localhost:3000
- You will see a message 'Hello World'

Source: https://nodejs.org/en/docs/guides/getting-started-guide/

Asynchronous programming – Node.js

- Node.js uses asynchronous programming (runs single-threaded, **non-blocking**) □ very memory efficient
- **Example**: Handling a file request:
  - In PHP/ASP.net:
    1. Sends the task to the computer's file system.
    2. Waits while the file system opens and reads the file.
    3. Returns the content to the client.
    4. Ready to handle the next request.
  - In Node.js:
    1. Sends the task to the computer's file system.
    2. Ready to handle the next request.
    3. When the file system has opened and read the file, the server returns the content to the client.

Blocking vs. non-blocking: PHP vs. Node.js

- PHP:

To select data from a table in MySQL, use the "SELECT" statement

Returns an array that corresponds to the fetched row

```php
<?php
$result = mysql_query('SELECT * FROM ...');
while($r = mysql_fetch_array($result)){
    // Do something
}

// Wait for query processing to finish...
?>
```

- Node.js:

Error handler

Callback!

```javascript
<script type="text/javascript">
mysql.query('SELECT * FROM ...', function (err, result, fields){
        // Do something
    });

// Don't wait, just continue executing
</script>
```

The third parameter of the callback function is an array containing information **about each field** in the **result object**

Blocking vs. non-blocking

- **Blocking**:
  - Read data from file `var data = fs.readFileSync( “test.txt” );`
  - Show data `console.log( data );`
  - Do other tasks `console.log( “Do other tasks” );`

- **Non-blocking:**
  - Read data from file
    - When read data completed, show data!
  - Do other tasks

Callback!

```
fs.readFile( “test.txt”, function( err, data ) {
console.log(data);
});
```

Using existing modules

```
var fs = require('fs');     // include File System module
var path = require('path');
// typically an object or a function is returned.

var buf = fs.readdir(process.argv[2], // command line arguments
  function(err, data) {
      for (i = 0; i < data.length; i++) {
          var s = path.extname(data[i]); // get the extension from a file path
          if (s === "." + process.argv[3]) {
              console.log(data[i]);
          }
       } // end of for
  } // end of callback function for readdir
```

The **process.argv** property returns an array containing the command line arguments passed when the Node.js process was launched.
The first element will be process.execPath.
The second element will be the path to the JavaScript file being executed.
The remaining elements will be any additional command line arguments.

”===“ vs. “==“

- By triple equals ”===“ we are testing for **strict equality**
  - Both the **type** and the **value** we are comparing have to be the same!
- By double equals “==“ we are testing for **loose equality**
  - Double equals also performs **type coercion**
  - **Type coercion**: two values are compared only **after** attempting to convert them into a common type
- Examples:
  - 5 === 5 // true
  - 77 === '77’ // false (Number v. String)
  - 77 == '77’ // true

Create your own modules – Example (1)

```
    exports.myDateTime = function () {
    return Date();
};
    var http = require('http');
var dt = require('./myfirstmodule');

http.createServer(function (req, res) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write("The date and time are currently: " + dt.myDateTime());
    res.end();
}).listen(8080);
```

Save the code in a file called "myfirstmodule.js"

Use **exports** to make properties and methods available outside the module

Include and use the module in any of your Node.js files.

Create your own modules – Example (2)

```
// FILE myModule.js
module.exports = function (dir, ext, callback) {
    var fs = require('fs');
    var path = require('path');
    var retValue =[];
    fs.readdir(dir, function(err, data) {{// data is an array of the names of the files in the directory excluding '.' and '..'
      if (err) return callback(err);
      retValue = data.filter(function(filename) {// filter the array of files with an extension extractor function
          return path.extname(filename) === "." + ext;
      });
     callback(null, retValue);
   }); // end of callback to readdir

 }; // end of function
```

Save the code in a file called " **myModule.js**"

USERS of this module will need to provide dir, extension, and callback

The path module provides one such function.

Create your own modules – Example (2)
Using the created module

```
var x = require('./mymodule');


// users need to provide dir, extension, callback


x(process.argv[2], process.argv[3], function(err, data)
 {
   if (err) return console.error ("error:", err);
     data.forEach(function(file) { // for each array element
       console.log (file);
     });
 } // end of callback function
); // end of call to x
```

Asynchronous I/O – Example

- **NO WAIT! until read is complete:**

```
var fs = require('fs');
var buf = fs.readFile(process.argv[2],
  function(err, data) { // callback
    if (err) { return console.log(err); }
    var sArray = data.toString().split("\n");
    console.log(sArray.length-1); // print number of lines
  } );
// No wait! – Do the next instruction right away!
```

Include the File System module:
fs = require('fs‘);
fs.**readFile**(file, [encoding], [callback]);

fs.**readFile**() method is used to read files .

Synchronous I/O – Example

Waits until i/o is done!

Example:

```
var fs = require('fs');  // node's modular code

var buf = fs.readFileSync(process.argv[2]);
// wait!!
var sArray = buf.toString().split("\n");
console.log(sArray.length-1); // print number of lines
```

Standard callback pattern

- **Callback function will look like:**

```
function (err, data) {
    if (err) { // handle error }
    else {
        // do something with data
    }
});
```

- This callback is **called once** when event happens/completes (for example, i/o is completed).

Literature – Node.js

- https://nodejs.org/en/
- https://www.w3schools.com/nodejs/default.asp
- https://www.tutorialspoint.com/nodejs/index.htm
- https://npmjs.org/