**Node.js**

* Node.js is a JavaScript run-time environment.
* *It is a JavaScript runtime built on*[*Chrome’s V8 JavaScript engine*](https://developers.google.com/v8/)*.*
* *It uses an event-driven, non-blocking I/O model that makes it lightweight and well-organized.*
* *Node.js’ package ecosystem is the largest ecosystem of open source libraries in the world which is [npm](https://www.npmjs.com/" \t "_blank).*
* Node.js is a free and an open source server environment
* It runs on different types of platforms such as Windows, Linux, Unix, Mac OS X.
* It uses JavaScript on the server
* The Node run-time environment contains everything a user may need in order to execute a program written in JavaScript.
* Node.js came became real when the original developers of JavaScript extended it from something a user could only run in a browser into something they could run on their device as a standalone application.
* Both your browser JavaScript and Node.js run on the V8 JavaScript runtime engine. This engine takes your JavaScript code and converts it into a faster machine code. Machine code is low level code which the computer can run without needing to first interpret it.

**Input/ Output (I/O)**

* It can be anything that ranges from reading/writing local files to making an HTTP request to an API.

**Blocking I/O (left) vs Non-Blocking I/O (right)**

* **Blocking I/O**
* In this method, the request of a user b’s data is not initiated until user a’s data is being displayed to the screen.
* **Non-Blocking I/O**
* A user can initiate a data request for user b without waiting for the response the request for user a. A user can initiate both requests in parallel.
* This non-blocking, I/O removes the need for multi-threading, since the server can handle multiple requests at the same time.

**File request using PHP or ASP**

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.

**File request using 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.

**Note:**

* Node.js removes the waiting, and continues with the next request.
* It runs single-threaded, non-blocking, asynchronously programming, which is very memory efficient.

**What can Node.js do?**

* It generates a dynamic page content
* Create, open, read, write, delete, and close files on the server
* It can collect form data
* Add, delete, modify data in your database

**What is a Node.js File?**

* It contains tasks that will be implemented on certain events
* Node.js files must be initiated on the server before having any effect
* Node.js files have extension ".js"

# Node.js Modules

## **What is a Module in Node.js?**

A set of functions a user desires to include in a application.

## **Built-in Modules**

* Contains a set of built-in modules which a user can use without installing it.

## **Include Modules**

* Uses  require() function with the name of the module:

var http = require('http');

## **Creating Own Modules**

* A user can create their own modules, and easily include them in their applications.
* It uses the exports keyword to make methods available outside the module file.

exports.myDateTime = function () {  
     return Date();  
 };

## **Including Own Module**

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);

# Node.js HTTP Module

## **The Built-in HTTP Module**

* Contains a built-in module called HTTP, which allows Node.js to move data over the Hyper Text Transfer Protocol (HTTP).
* It uses the require() method to add a HTTP module:

var http = require('http');

## **Node.js as a Web Server**

* The HTTP module can make an HTTP server that listens to server ports and provides a response back to the client.
* Use the createServer() method to create an HTTP server:

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

## **Adding HTTP Header**

If the response from the HTTP server is meant to be shown as HTML, the user should comprise an HTTP header with the correct content type:

var http = require('http');  
 http.createServer(function (req, res) {  
**res.writeHead(200, {'Content-Type': 'text/html'});**  res.write('Hello World!');  
  res.end();  
 }).listen(8080);

## **Reading Query String**

The function passed into the http.createServer() has a req argument that represents the request from the client, as an object (http.IncomingMessage object).

This object has a property called "url" which holds the part of the url that comes after the domain name:

demo\_http\_url.js

var http = require('http');  
 http.createServer(function (**req**, res) {  
     res.writeHead(200, {'Content-Type': 'text/html'});  
    res.write(**req.url**);  
     res.end();  
 }).listen(8080);

## **Splitting Query String**

There are built-in modules to split the query string into readable parts, such as the URL module.

var http = require('http');  
 **var url = require('url');**  
 http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
 **var q = url.parse(req.url, true).query;**   var txt = **q.year** + " " + **q.month**;  
   res.end(txt);  
 }).listen(8080);

# Node.js File System Module

## **Node.js as a File Server**

* The Node.js file system module let a user work with the file system on a computer.
* Uses the require() method to include the File System module

var fs = require('fs');

**Common use for the File System module:**

* Read files
* Create files
* Update files
* Delete files
* Rename files

## **Read Files**

The fs.readFile() method is used to read files on your computer.

var http = require('http');  
 var fs = require('fs');  
 http.createServer(function (req, res) {  
 **fs.readFile('demofile1.html', function(err, data) {**     res.writeHead(200, {'Content-Type': 'text/html'});  
     res.write(data);  
     res.end();  
  });  
 }).listen(8080);

## **Create Files**

The File System module has methods for creating new files:

* fs.appendFile()
* fs.open()
* fs.writeFile()

The fs.appendFile() method appends specified content to a file. If the file does not exist, the file will be created:

var fs = require('fs');  
  
 fs.appendFile('mynewfile1.txt', 'Hello content!', function (err) {  
   if (err) throw err;  
   console.log('Saved!');  
 });

The fs.open() method gest a "flag" as the second argument, if the flag is "w" for "writing", the specified file is opened for writing. If the file does not exist, an empty file is created:

var fs = require('fs');  
  
 fs.open('mynewfile2.txt', 'w', function (err, file) {  
   if (err) throw err;  
   console.log('Saved!');  
 });

The fs.writeFile() method exchanges the specific file and content if it exists. If the file does not exist, a new file, containing the specified content, will be created:

var fs = require('fs');  
  
 fs.writeFile('mynewfile3.txt', 'Hello content!', function (err) {  
   if (err) throw err;  
   console.log('Saved!');  
 });

## **Update Files**

The File System module has methods for updating files:

* fs.appendFile()
* fs.writeFile()

The fs.appendFile() method appends the specified content at the end of the specified file:

var fs = require('fs');  
  
 fs.appendFile('mynewfile1.txt', ' This is my text.', function (err) {  
   if (err) throw err;  
   console.log('Updated!');  
 });

The fs.writeFile() method takes place the specified file and content:

var fs = require('fs');  
  
 fs.writeFile('mynewfile3.txt', 'This is my text', function (err) {  
   if (err) throw err;  
   console.log('Replaced!');  
 });

## **Delete Files**

Use the fs.unlink() method to delete a file with the File System module

The fs.unlink() method deletes the specified file:

var fs = require('fs');  
  
 fs.unlink('mynewfile2.txt', function (err) {  
   if (err) throw err;  
   console.log('File deleted!');  
 });

## **Rename Files**

Use the fs.rename() method to rename a file with the File System module

The fs.rename() method renames the specified file:

var fs = require('fs');  
  
 fs.rename('mynewfile1.txt', 'myrenamedfile.txt', function (err) {  
   if (err) throw err;  
   console.log('File Renamed!');  
 });

# Node.js Upload Files

## **The Formidable Module**

There is a very good module for working with file uploads, called "Formidable".

The Formidable module can be downloaded and installed using NPM:

C:\Users\Your Name>npm install formidable

After you have downloaded the Formidable module, you can include the module in any application:

var formidable = require('formidable');

## **Upload Files**

## **Creating an Upload Form**

Create a Node.js file that writes an HTML form, with an upload field:

var http = require('http');  
  
 http.createServer(function (req, res) {  
   res.writeHead(200, {'Content-Type': 'text/html'});  
   res.write('<form action="fileupload" method="post" enctype="multipart/form-data">');  
   res.write('<input type="file" name="filetoupload"><br>');  
   res.write('<input type="submit">');  
   res.write('</form>');  
   return res.end();  
 }).listen(8080);

## **Parsing the Uploaded File**

Include the Formidable module to be able to parse the uploaded file once it reaches the server.

When the file is uploaded and parsed, it gets placed on a temporary folder on your computer.

var http = require('http');  
 **var formidable = require('formidable');**  
 http.createServer(function (req, res) {  
**if (req.url == '/fileupload') {  
     var form = new formidable.IncomingForm();  
     form.parse(req, function (err, fields, files) {  
       res.write('File uploaded');  
       res.end();  
     });**   } else {  
    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.write('<form action="fileupload" method="post" enctype="multipart/form-data">');  
    res.write('<input type="file" name="filetoupload"><br>');  
    res.write('<input type="submit">');  
    res.write('</form>');  
    return res.end();  
   }  
 }).listen(8080);

## **Save the File**

* When a file is successfully uploaded to the server, it is located on a temporary folder.
* The path to this directory can be found in the "files" object, passed as the third argument in the parse() method's callback function.
* To move the file to the folder of your choice, use the File System module, and rename the file:

var http = require('http');  
 var formidable = require('formidable');  
 **var fs = require('fs');**  
 http.createServer(function (req, res) {  
   if (req.url == '/fileupload') {  
     var form = new formidable.IncomingForm();  
     form.parse(req, function (err, fields, files) {  
**var oldpath = files.filetoupload.path;  
       var newpath = 'C:/Users/Your Name/' + files.filetoupload.name;  
       fs.rename(oldpath, newpath, function (err) {  
         if (err) throw err;  
         res.write('File uploaded and moved!');  
        res.end();  
       });**  });  
   } else {  
    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.write('<form action="fileupload" method="post" enctype="multipart/form-data">');  
    res.write('<input type="file" name="filetoupload"><br>');  
     res.write('<input type="submit">');  
     res.write('</form>');  
     return res.end();  
   }  
 }).listen(8080);

# Node.js Events

## **Events in Node.js**

* Objects in Node.js can fire events, like the readStream object fires events when opening and closing a file:

var fs = require('fs');  
 var rs = fs.createReadStream('./demofile.txt');  
 rs.on('open', function () {  
  console.log('The file is open');  
 });

## **Events Module**

* Node.js has a built-in module, called "Events", where you can create-, fire-, and listen for- your own events.
* Use the require() method to include the built-in Events module.
* All event properties and methods are an instance of an EventEmitter object.
* To be able to access these properties and methods, create an EventEmitter object:

var events = require('events');  
 var eventEmitter = new events.EventEmitter();

## **The EventEmitter Object**

* You can assign event handlers to your own events with the EventEmitter object.
* To fire an event, use the emit() method.

var events = require('events');  
 var eventEmitter = new events.EventEmitter();  
  
 //Create an event handler:  
 var myEventHandler = function () {  
   console.log('I hear a scream!');  
 }  
  
 //Assign the event handler to an event:  
 eventEmitter.on('scream', myEventHandler);  
  
 //Fire the 'scream' event:  
 eventEmitter.emit('scream');