What is Node.js?

* Node.js is an open source server environment
* Node.js is free
* Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
* Node.js uses JavaScript on the server

Here is how PHP or ASP handles a file request:

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.

Here is how Node.js handles a file request:

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.

Node.js eliminates the waiting, and simply continues with the next request.

Node.js runs single-threaded, non-blocking, asynchronously programming, which is very memory efficient.

What Can Node.js Do?

* Node.js can generate dynamic page content
* Node.js can create, open, read, write, delete, and close files on the server
* Node.js can collect form data
* Node.js can add, delete, modify data in your database

What is a Node.js File?

* Node.js files contain tasks that will be executed on certain events
* A typical event is someone trying to access a port on the server
* Node.js files must be initiated on the server before having any effect
* Node.js files have extension ".js"

Create Server:

var http = require('http');  
  
http.createServer(function (req, res) {

console.log(“hi”);

    res.writeHead(200, {'Content-Type': 'text/plain'});  
    res.end('First Page!');

}).listen(8080);

The first argument of the res.writeHead() method is the status code, 200 means that all is OK, the second argument is an object containing the response headers.

Run Node file:

$node filename.js

## What is a Module in Node.js?

Consider modules to be the same as JavaScript libraries.

A set of functions you want to include in your application.

## Built-in Modules

Node.js has a set of built-in modules which you can use without any further installation.

## Include Modules

To include a module, use the require() function with the name of the module:

var http = require('http');

## Create Your Own Modules

You can create your own modules, and easily include them in your applications.

The following example creates a module that returns a date and time object:

### **Example**

Create a module that returns the current date and time:

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

Use the exports keyword to make properties and methods available outside the module file.

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

## Include Your Own Module

Now you can include and use the module in any of your Node.js files.

### **Example**

Use the module "myfirstmodule" in a Node.js file:

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);

## Read the 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);

## Split the Query String

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

### **Example**

Split the query string into readable parts:

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);

The address:

<http://localhost:8080/?year=2017&month=July>

Will produce this result:

2017 July

Email Example:

## The Nodemailer Module

The Nodemailer module makes it easy to send emails from your computer.

$npm install nodemailer

var nodemailer = require('nodemailer');  
  
var transporter = nodemailer.createTransport({  
  service: 'gmail',  
  auth: {  
    user: 'youremail@gmail.com',  
    pass: 'yourpassword'  
  }  
});  
  
var mailOptions = {  
  from: 'youremail@gmail.com',  
  to: 'myfriend@yahoo.com',  
  subject: 'Sending Email using Node.js',  
  text: 'That was easy!'  
};  
  
transporter.sendMail(mailOptions, function(error, info){  
  if (error) {  
    console.log(error);  
  } else {  
    console.log('Email sent: ' + info.response);  
  }  
});

Node with My SQL server:

## Install MySQL Driver

Once you have MySQL up and running on your computer, you can access it by using Node.js.

To access a MySQL database with Node.js, you need a MySQL driver. This tutorial will use the "mysql" module, downloaded from NPM.

$npm install mysql

var mysql = require('mysql');  
  
var con = mysql.createConnection({  
  host: "localhost",  
  user: "yourusername",  
  password: "yourpassword"  
});  
  
con.connect(function(err) {  
  if (err) throw err;  
  console.log("Connected!");

con.query("**CREATE DATABASE mydb**", function (err, result) {  
    if (err) throw err;  
    console.log("Database created");  
  });

});

Create table

con.connect(function(err) {  
  if (err) throw err;  
  console.log("Connected!");  
  var sql = "**CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255)**)";  
  con.query(sql, function (err, result) {  
    if (err) throw err;  
    console.log("Table created");  
  });  
});

Insert data:

con.connect(function(err) {  
  if (err) throw err;  
  console.log("Connected!");  
**var sql = "INSERT INTO customers (name, address) VALUES ('Company Inc', 'Highway 37')";**  con.query(sql, function (err, result) {  
    if (err) throw err;  
    console.log("1 record inserted");  
  });  
});

Select:

con.connect(function(err) {  
  if (err) throw err;  
  con.query("**SELECT \* FROM customers**", function (err, result, fields) {  
    if (err) throw err;  
    console.log(result);  
  });  
});

Update:

con.connect(function(err) {  
  if (err) throw err;  
**var sql = "UPDATE customers SET address = 'Canyon 123' WHERE address = 'Valley 345'";**  con.query(**sql,** function (err, result) {  
    if (err) throw err;  
    console.log(result.affectedRows + " record(s) updated");  
  });  
});

MongoDB

## Install MongoDB Driver

Let us try to access a MongDB database with Node.js.

To download and install the official MongoDB driver, open the Command Terminal and execute the following:

Download and install mongodb package:

$npm install mongodb

var mongo = require('mongodb');

## Creating a Database

To create a database in MongoDB, start by creating a MongoClient object, then specify a connection URL with the correct ip address and the name of the database you want to create.

MongoDB will create the database if it does not exist, and make a connection to it.

### **Example**

Create a database called "mydb":

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/mydb";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  console.log("Database created!");  
  db.close();  
});

## Creating a Collection

To create a collection in MongoDB, use the createCollection() method:

### **Example**

Create a collection called "customers":

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  dbo.createCollection("customers", function(err, res) {  
    if (err) throw err;  
    console.log("Collection created!");  
    db.close();  
  });  
});

## Insert Into Collection

To insert a record, or document as it is called in MongoDB, into a collection, we use the insertOne() method.

A **document** in MongoDB is the same as a **record** in MySQL

The first parameter of the insertOne() method is an object containing the name(s) and value(s) of each field in the document you want to insert.

It also takes a callback function where you can work with any errors, or the result of the insertion:

### **Example**

Insert a document in the "customers" collection:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  var myobj = { name: "Company Inc", address: "Highway 37" };  
  dbo.collection("customers").insertOne(myobj, function(err, res) {  
    if (err) throw err;  
    console.log("1 document inserted");  
    db.close();  
  });  
});

## Find One

To select data from a collection in MongoDB, we can use the findOne() method.

The findOne() method returns the first occurrence in the selection.

The first parameter of the findOne() method is a query object. In this example we use an empty query object, which selects all documents in a collection (but returns only the first document).

### **Example**

Find the first document in the customers collection:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  dbo.collection("customers").findOne({}, function(err, result) {  
    if (err) throw err;  
    console.log(result.name);  
    db.close();  
  });  
});

## Filter the Result

When finding documents in a collection, you can filter the result by using a query object.

The first argument of the find() method is a query object, and is used to limit the search.

### **Example**

Find documents with the address "Park Lane 38":

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
**var query = { address: "Park Lane 38" };**  dbo.collection("customers").find(**query**).toArray(function(err, result) {  
    if (err) throw err;  
    console.log(result);  
    db.close();  
  });  
});

## Filter With Regular Expressions

You can write regular expressions to find exactly what you are searching for.

**Regular expressions can only be used to query strings.**

To find only the documents where the "address" field starts with the letter "S", use the regular expression /^S/:

### **Example**

Find documents where the address starts with the letter "S":

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  var query = { address: **/^S/** };  
  dbo.collection("customers").find(query).toArray(function(err, result) {  
    if (err) throw err;  
    console.log(result);  
    db.close();  
  });  
});

## Sort the Result

Use the sort() method to sort the result in ascending or descending order.

The sort() method takes one parameter, an object defining the sorting order.

### **Example**

Sort the result alphabetically by name:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
**var mysort = { name: 1 };**  dbo.collection("customers").find()**.sort(mysort)**.toArray(function(err, result) {  
    if (err) throw err;  
    console.log(result);  
    db.close();  
  });  
});

## Delete Document

To delete a record, or document as it is called in MongoDB, we use the deleteOne() method.

The first parameter of the deleteOne() method is a query object defining which document to delete.

**Note:** If the query finds more than one document, only the first occurrence is deleted.

### **Example**

Delete the document with the address "Mountain 21":

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
**var myquery = { address: 'Mountain 21' };**  dbo.collection("customers").deleteOne(myquery, function(err, obj) {  
    if (err) throw err;  
    console.log("1 document deleted");  
    db.close();  
  });  
});

## Delete Many

To delete more than one document, use the deleteMany() method.

The first parameter of the deleteMany() method is a query object defining which documents to delete.

### **Example**

Delete all documents were the address starts with the letter "O":

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
**var myquery = { address: /^O/ };**  dbo.collection("customers").deleteMany(myquery, function(err, obj) {  
    if (err) throw err;  
    console.log(obj.result.n + " document(s) deleted");  
    db.close();  
  });  
});

## Drop Collection

You can delete a table, or collection as it is called in MongoDB, by using the drop() method.

The drop() method takes a callback function containing the error object and the result parameter which returns true if the collection was dropped successfully, otherwise it returns false.

### **Example**

Delete the "customers" table:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  dbo.collection("customers").drop(function(err, delOK) {  
    if (err) throw err;  
    if (delOK) console.log("Collection deleted");  
    db.close();  
  });  
});

## db.dropCollection

You can also use the dropCollection() method to delete a table (collection).

The dropCollection() method takes two parameters: the name of the collection and a callback function.

### **Example**

Delete the "customers" collection, using dropCollection():

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb"); **dbo.dropCollection("customers", function(err, delOK) {**    if (err) throw err;  
    if (delOK) console.log("Collection deleted");  
    db.close();  
  });  
});

## Update Document

You can update a record, or document as it is called in MongoDB, by using the updateOne() method.

The first parameter of the updateOne() method is a query object defining which document to update.

**Note:** If the query finds more than one record, only the first occurrence is updated.

The second parameter is an object defining the new values of the document.

### **Example**

Update the document with the address "Valley 345" to name="Mickey" and address="Canyon 123":

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://127.0.0.1:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  var myquery = { address: "Valley 345" };  
  var newvalues = { $set: {name: "Mickey", address: "Canyon 123" } };  
  dbo.collection("customers").updateOne(myquery, newvalues, function(err, res) {  
    if (err) throw err;  
    console.log("1 document updated");  
    db.close();  
  });  
});

Limit

### **Example**

Limit the result to only return 5 documents:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://localhost:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  dbo.collection("customers").find()**.limit(5)**.toArray(function(err, result) {  
    if (err) throw err;  
    console.log(result);  
    db.close();  
  });  
});

Join:

### **Example**

Join the matching "products" document(s) to the "orders" collection:

var MongoClient = require('mongodb').MongoClient;  
var url = "mongodb://127.0.0.1:27017/";  
  
MongoClient.connect(url, function(err, db) {  
  if (err) throw err;  
  var dbo = db.db("mydb");  
  dbo.collection('orders').aggregate([  
**{ $lookup:  
       {  
         from: 'products',  
         localField: 'product\_id',  
         foreignField: '\_id',  
         as: 'orderdetails'  
       }  
     }**    ]).toArray(function(err, res) {  
    if (err) throw err;  
    console.log(JSON.stringify(res));  
    db.close();  
  });  
});