**ASSIGNMENT OF WEEK 1 (FULL STACK DEVELOPMENT)**

**FOR ADVANCED WEB DEVELOPMENT**

Ques No.2: Explain in detail about Node.js.

Answer 2:

**Node.js: The web server/platform**

* 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
* Example: given below
* var http = require('http');  
    
  http.createServer(function (req, res) {  
    res.writeHead(200, {'Content-Type': 'text/plain'});  
    res.end('Hello World!');  
  }).listen(8080);
* Node.js uses asynchronous programming!
* 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

**Initiate the Node.js File**

* Start command line interface, write node myfirst.js and hit enter:
* Initiate "myfirst.js":
* C:\Users\our Name>node myfirst.js
  + **Node.js Modules**
* Consider modules to be the same as JavaScript libraries.
* A set of functions we want to include in our application.

**INCLUDE Modules**

* To include a module, use the require() function with the name of the module:
* var http = require('http');
* Now our application has access to the HTTP module, and is able to create a server:
* http.createServer(function (req, res) {  
    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.end('Hello World!');  
  }).listen(8080);

**CREATE OUR OWN MODULES:**

* We can create our own modules, and easily include them in our applications.
* Example: Create a module that returns the current date and time:

exports.myDateTime =function(){

returnDate();  
};

* 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 include and use the module in any Node.js files.
* Example: the module "myfirstmodule" in a Node.js file:

var http = require('http');  
var**dt = require(**'./myfirstmodule'**);          ./ to locate the module**  
  
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**
* Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).
* To include the HTTP module, use the require() method:

var http = require('http');

* **Node.js as a Web Server**
* The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.
* Use the createServer() method to create an HTTP server:
* Example

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

* **Add an HTTP Header**
* If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type:
* Example

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

**Node.js File System Module**

* The Node.js file system module allows to work with the file system on computer.
* To include the File System module, use the require() method:

var fs = require('fs');

* **Read Files**
* The fs.readFile() method is used to read files on computer.
* Assume we have the following HTML file (located in the same folder as Node.js):
* Create a Node.js file that reads the HTML file, and return the content:
* Example

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);  
    return res.end();  
  });  
}).listen(8080);

* **Create Files**
* The File System module has methods for creating new files:
* fs.appendFile()

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

* fs.open()

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

* fs.writeFile()

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()

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

* fs.writeFile()

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

* **Delete Files**
* To delete a file with the File System module,  use the fs.unlink() method.

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

* **Rename Files**
* To rename a file with the File System module,  use the fs.rename() method.
* var fs = require('fs');  
    
  fs.rename('mynewfile1.txt', 'myrenamedfile.txt', function (err) {  
    if (err) throw err;  
    console.log('File Renamed!');  
  });

**Node.js URL Module**

* The URL module splits up a web address into readable parts.
* To include the URL module, use the require() method:

var url = require('url');

**Node.js NPM**

* A package manager, npm, gets installed when we install Node.js. npm gives the ability to download Node.js modules or packages to extend the functionality of your application.
* Currently, more than 350,000 packages are available through npm, an indication of how much depth of knowledge and experience can bring to an application.

Ques No.3: Define Full Stack Development?

Answer 3:  Full-stack development, is developing all parts of a website or application. The full stack starts with the database and web server in the back end, contains application logic and control in the middle, and goes all the way through to the user interface at the front end.

The MEAN stack is a pure JavaScript stack comprised of four main technologies, with a cast of supporting technologies:

* + MongoDB—the database
  + Express—the web framework
  + Angular—the front-end framework
  + Node.js—the web server

Ques No.4: Why learn the Full Stack Development?

Answer 4:

* + We are more likely to have a better view of the bigger picture by understanding the different areas and how they fit together.
  + You’ll form an appreciation of what other parts of the team are doing and what they need to be successful.
  + Like other team members, you can move around more freely.
  + We can build applications end-to-end by ourself without depending on other people.
  + We develop more skills, services, and capabilities to offer customers.
  + Diverse skill.
  + High Demand.
  + Great Pay.
  + Creative flexibility.
  + Better Productivity.