**Chapter 3**

**Node JS – Http Server**

**Node JS Server**

In this chapter, we will be creating our own node http server to serve files and return responses.

**Http**

Before we move on to the creation of our node js server, let’s do a brief introduction to Hypertext Transfer Protocol(Http), the most popular application layer protocol used for communicating between the client and the server.

It provides a standardized way for computers to communicate with each other and specifies how clients' request data will be constructed and sent to the server, and how the servers respond to these requests.

To communicate with the server, the client must indicate the HTTP action such as GET, POST, PUT, DELETE, TRACE,OPTIONS, CONNECT. We will focus on GET, PUT, POST, and DELETE which is used when we create our restful web services. In the HTTP protocol, the request message typically carries information such as the action and target URL in the request message. You will also specify the server url you are communicating with. When the server receives your request, the server will retrieve the data from its data storage, package this data in an appropriate format and send the data back to the client. The data format is typically in html/css/javascript or JSON format for our restful web services.

|  |
| --- |
| REQUEST GET /index.html HTTP/1.1  Host: www.abc.com    RESPONSE HTTP/1.1 200 OK  Content-Type: text/html <html>….</html> |

A HTTP Request sent to the server will typically follow the below format:

|  |
| --- |
|  |

The Http Response returned by the server will also follow the corresponding below format:

|  |
| --- |
|  |

Below table shows the common Http response codes.

|  |
| --- |
| Code and Meaning |
| 200 - OK |
| 201 - Created |
| 301 - Moved Permanently |
| 304 - Not Modified |
| 400 - Bad Request |
| 401 - Unauthorized |
| 403 - Forbidden |
| 404 - Not Found |
| 422 - Unprocessable Entry |
| 500 - Internal Server Error |
| 505 - HTTP Version Not Supported |

As an example, consider a typical HTTP request message sent to the server from the client that requests for contents of the index.html page. A request like this would typically have an empty body in the request message. Most of the information will be encoded in the request line plus the headers of the request message. When the client sends the request to the server. The server will process the request and then send back a reply to the client side. The reply message is organized into three parts, a status line, the headers and the actual body of the message. In this case, assuming the file is successfully retrieve, the response message will return with a status code 200. The header will contain information such as the data format encoding of the returned message data. Then the body will contain the body of the message, which is the HTML code for the index.html page for this example.

Besides html code, the server may also encode the data in a specific format like JavaScript Object Notation(JSON) format. In this module, we will be dealing with data that is encoded mainly in JSON. Most client side applications including both web and mobile applications these days typically communicate with the web server and the data exchange format is usually JSON. Examples of web apis that return JSON format include LTA Datamall and Alpha Vantage.

**What is JSON**

JSON stands for JavaScript Object Notation.

**Features of JSON**:

* It is light-weight
* It is language independent
* Easy to read and write
* Text based, human readable data exchange format

A single json object has key-value pairs describing the object:

{

"name":"John",

"age":"18",

"course":"DIT"

}

To represent multiple json objects, we have a json array:

{"students":[

{"name":"John","age":"18","course":"DIT"},

{"name":"Mary","age":"18","course":"DISM"},

{"name":"Jack","age":"28","course":"NSFSW"}

]

}

We can directly use the JSON object by callinig the name of the variable object and then the property.

var student1={

"name":"John",

"age":"18",

"course":"DIT"

};

console.log(student1.name);

console.log(student1.age);

console.log(student1.course);

To convert the javascript object back into json, we can use stringify function

var jsonStudent=JSON.stringify(jsStudent);

**Our first Node Server App**

Now that we have knowledge about the basics of HTTP protocol in web communication, let’s look at how Node can be used to construct an HTTP server.

To build our own web server, we will look at a few Javascript modules that is built into Node JS:

1. http module: provides a high-performing foundation for an HTTP stack
2. Path module: provides resolving of full paths, check of file paths etc
3. File module: provides file reading, checking purposes

**Http Module**

To make use of the various modules, we just need to import them into the module using the require function that we learnt in the previous chapter. To use the http module for example, we can include the statement const HTTP = require(‘http’); in the code. Once the module is imported, the HTTP module supports a createServer function which takes, as a parameter, a callback function that has two parameters: request and response:

1. The request parameter is the request message that comes in from the client side and we can process and extract information from the request message.
2. The response parameter is the response message which we can construct the various header values and also the body of the HTTP response message.

For the request message object, we can extract information from like the header and the body information. Useful information like the URL and the method of the request message can be extract. For the response returned to the client, you can construct the response object by setting information like the header, the status code and message body. The message body may be standard text, HTML or other data types like JSON.

Once the server is created, we can start the server and make it listen to requests on a certain port.

**Node Path Module**

The path module can be imported using the statement const path = require(‘path’);. The path module enables us to examine whether a file exists or examine more details about a file, such as the file extension. It also has function which will allow us to convert a relative path into an absolute for a file.

**File Module**

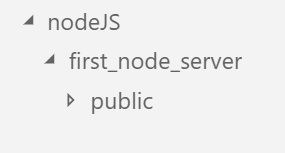
The file module will enable us to read and write files on the computer. It can also be used to check whether a file exists. In order to host files on your server, you need to read the file and encode the file data in the response message.

**Our first Node Http Server**

Now, that we understand some details about the HTTP server and how it can be constructed, let’s construct a simple Node HTTP server which will serve up some basic information and some html files that exist in a public folder in our project.

First create a subfolder named firstNodeSever in the NodeJS parent folder.

In the first\_node\_server folder, create a subfolder named public.



Startup the terminal in visual studio code and type the following to initialize a package.json file in the first\_node\_server folder:

|  |
| --- |
| npm init |

Accept all default suggestions and you should end up with a package.json similar to the below:

|  |
| --- |
| {  "name": "first\_node\_server",  "version": "1.0.0",  "description": "",  "main": "index.js",  "scripts": {  "test": "echo \"Error: no test specified\" && exit 1"  },  "author": "",  "license": "ISC"  } |

The json file contains some information about the project and subsequently is also used by node to maintain info such as additional modules downloaded to the project. We will see the usefulness of this file in the next chapter.

Create a file named server.js and save this in first\_node\_server folder:

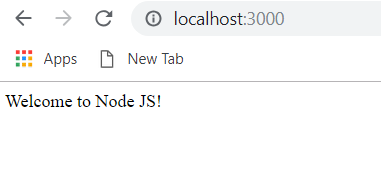
|  |
| --- |
| const http = require('http');  const hostname = 'localhost';  const port = 3000;  const server = http.createServer((req, res) => {  res.statusCode = 200;  res.setHeader('Content-Type', 'text/html');  res.end('<html><body>Welcome to Node JS!</body></html>');  });  server.listen(port, hostname, () => {  console.log(`Server started and accessible via http://${hostname}:${port}/`);  }); |

Start the server by typing the following in the terminal:

|  |
| --- |
| node server.js |

Try accessing the webpage from your browser using the url <http://localhost:3000/>

You should see the follow output:



**Serving files in your server**

Now, let’s extend the code to support ONLY get method and also serve files to the client.

First add 2 more files to the public subfolder in your project.

index.html:

|  |
| --- |
| <html>  <body>  Welcome to Node JS!!  </body>  </html> |

error.html

|  |
| --- |
| <html>  <body>  File does not exist or web server does not support post method!  </body>  </html> |

Modify server.js using the below given code as a guide:

|  |
| --- |
| …  const fs = require('fs');  const path = require('path');  …  const server = http.createServer((req, res) => {  console.log('Request for page ' + req.url + ' using ' + req.method + ‘method’);  if (req.method == 'GET') {  var fileUrl=req.url;  if (req.url == '/') //default file is index.html  fileUrl = '/index.html';    var filePath = path.resolve('./public'+fileUrl);  fs.exists(filePath, (exists) => {  if (!exists) {  fileUrl='/error.html';  filePath = path.resolve('./public'+fileUrl);    }else{  res.statusCode = 200;  res.setHeader('Content-Type', 'text/html');  }  fs.createReadStream(filePath).pipe(res);  });  }  else {  fileUrl='/error.html';  filePath = path.resolve('./public'+fileUrl);  fs.createReadStream(filePath).pipe(res);  }  });  … |

When you request for a file index.html with the following url <http://localhost:3000/index.html>, you would notice that the “Welcome to Node JS!!” message is displayed in the web page.

When a file that is not present is requested, such as with url <http://localhost:3000/a.html>, “File does not exist or web server does not support post method!” message would be displayed in the web page.

From the messages displayed, what do you think is the method of the request when you type in the url in your browser and the request is sent to the server?

We will be exploring in more details the difference of the request methods get, post, put, delete in the next chapter when we look at Restful Web Services.