Writing Our First Server

Server: What & How?

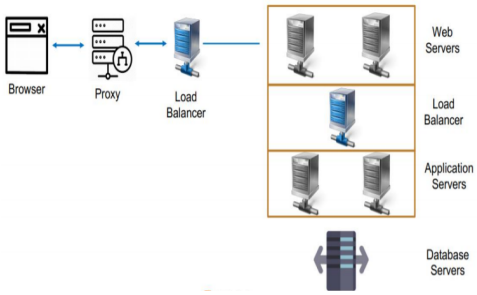
A Server is something that serves the functionality of another computer system, device, or any program that is known as a client.

In technical terms it can be defined as - “Web server can refer to either hardware or the software that helps to deliver web content [i.e the functionalities] that can be accessed through the internet”.

Request-Response Cycle - There is a request-response cycle between the system and the server. The system sends the request to the server then the system opens a TCP/IP connection {Whenever communication takes place between two devices they need to follow some protocols } via socket and request for a document, till then the server waits for the reply, and once it gets the reply, the socket is closed.

Client-Server Architecture -

WEB ARCHITECTURE (Scale-Up) -



Setting Up Our First Node.js Server

- Key Points while starting the server.
- Create the folder (basichttpserver).
- Create the first file in the directory (i.e index.js). It is also called the entry point in our code directory.
- Start the server using "npm init", which gives information about your project to the outer world. It comprises the following information -: package name, version, description, entry point, test command, git repository, keywords, author, license. It also contains confirmation about all the details that you have mentioned. This will create a package.json file which will have all the information that is mentioned above in key-value pairs format. Package.json shows the configuration of your server to the outer world.
- Here we are using an HTTP module that gives the functionality to run our server and looks into our file system and whenever a user requests a file this module sends the response accordingly.
- To use HTTP server and client one must require it and also set the port on which we want to display our project.
- To create a server one has to call the create server function. We also make a function that can handle the error if present. In case an error is not present, this function will redirect us to the desired page.
- Use the 'node file.js' command to run your server. In the browser, we go to the URL - localhost:port_no.

    - **Note**: *A port number can run a single server at a time and we can run multiple node.js servers on a single machine*

Serving A Response to the Browser

- Till now we are just sending the request. No response is seen, the server is just loading, So now, we will send the response. For that, we have to make a request handler that takes two arguments by default. One is requested and another one is a response, and add this function to our server that is created and print the content that we want to display on the screen.
- We can display different URLs on our terminal by using console.log in our request handler function. Using req.URL we get to know what URL is being requested by the user.

    - **Note**: *If you make any changes in the code you have to restart the server again.*

Understanding Responses :

- **Note:** *For different URL requests we are sending the same response which is not the correct way to depict the real-world scenarios we don't get the same response for different URLs.*

Serving HTML!

- We have to set our response code as a first argument in our res.writeHead which tells the status of our response based on our request or any error on the server-side using a numeric code.

  Some of them are - :

  404 - Not Found,
  200 - success,
  500 - Internal Server error,
  422 - Invalid Input
  403 - the client does not have access rights to the content

- Along with response code in our writeHead we have to pass one more argument to it - which is a set of options, the first one would be content-type which can be text/HTML [We give two options usually because this is the protocol defined & also some browser may not be able to render everything].
- In the Header, you tell what is coming on, what type of content is coming on, or any other hidden information that you want to send from the server to the browser.

```
function requestHandler(req,res) {
    console.log(req.url);
    res.writeHead(200,{'content-type' : 'text/html'});
    res.end('<h1>Gotcha!!</h1>');
}

const server = http.createServer(requestHandler);
```

Reading & Serving HTML from a File -

- It is generally not a good way to send all our HTML files in the above format so what one should do is to make a separate file of Html which will tell Node.js to read that file and return the data that it gets depending upon whatever URL is requested.
- Create an HTML file [index.html] and write a basic layout of HTML along with internal CSS [Resides inside the same page]
- To read the Index.js file, we need something that can read or write into or from files, there is a module that is built-in node.js that is fs.
- Use fs to read the Html file[i.e index.html] that we have created.
- To read the file which is built in asynchronous function, we need a path in which the file resides. This path needs to be sent to the function as a parameter. The first argument is used to send the path to the asynchronous function {./ - Used when we want a file that is on the same level}. Along with that, there is one call back function which will take two arguments
    - one is an error, and
    - another one is the data that we received while reading the file.
- In case of an error, it will render the error message that we want to display, and if not then our data will be displayed on the screen.

```
const fs = require('fs');
function requestHandler(req,res) {
    console.log(req.url);
    res.writeHead(200,{'content-type' : 'text/html'});
    fs.readFile('./index.html',function(err,data) {
        if(err) {
            console.log('error',err);
            return res.end(<h1>Error!!</h1>);
        }
        return res.end(data);
    });
}

const server = http.createServer(requestHandler);
```

Nodemon, Automatic Server Restarts -

- The pain of starting the server again and again whenever any js file has been changed is now removed by installing a node package using npm . The package used for this is nodemon.
- We can install nodemon using a command: npm install -g nodemon [-g stands for globally to install nodemon package globally].
- As soon as the JS file changes it restarts the server automatically.
- The command to run the file on the terminal is “nodemon index.js”.

- Whenever an HTML file is changed it does not restart the server again as that file is being read by the file system and then displayed.
- Whereas, the JS file is compiled at once and all the functions are loaded into the memory. Then those functions are executed as per the request.

Extending Multiple Pages: -

- Here we are going to be rendering multiple HTML files for different requests, earlier we were rendering the same HTML file for all requests.
- Creating two files one is 404.html and one more profile.html and based on URLs we render them.
- Provide basic styling and content to both pages.
- When we type localhost:8000/profile - {We get the data of profile.html} other than that any URL will render the data of 404.html.
- Using switch cases we can render multiple pages based on multiple URLs that the user wants to and scale up our website.

Assignment - :

- ***Serve your static websites using a node.js server by dividing them into different files and sections.***

Note: ***Package.lock.json has a complete history of what all dependencies have been required by the project. A complete tree of dependencies & sub dependencies that are there, which is very useful when you want to track your project.***

- ***Package.json contains dependencies that contain different libraries that are used in your project.***

Summary:-

- SetUp our directory structure where we can code.
- Using HTML and File System module we run our server
- Render a plain text.
- Set the headers and then render an H1 tag using that.
- We created different URLs using the request and response cycle.
- Using req. URL, we get to know what URL is being requested by the user.
- Render a separate HTML file to the browser along with some styling.
- Install our first package using npm.
- Render Multiple HTML pages for the different request of URLs along with the same page CSS styling using switch cases