Lab 14: *NodeJS* Backend

(http) Web Server

Node JS + NPM

IMPORTANT NOTE: Install Node JS

Table of Content: *Implementing a Simple (HTTP) Web Server*

Lab Introduction

Prerequisites

None. Must have installed NodeJS.

Motivation

Understand how a web server listens for and responds to a browser's HTTP requests.

Goal

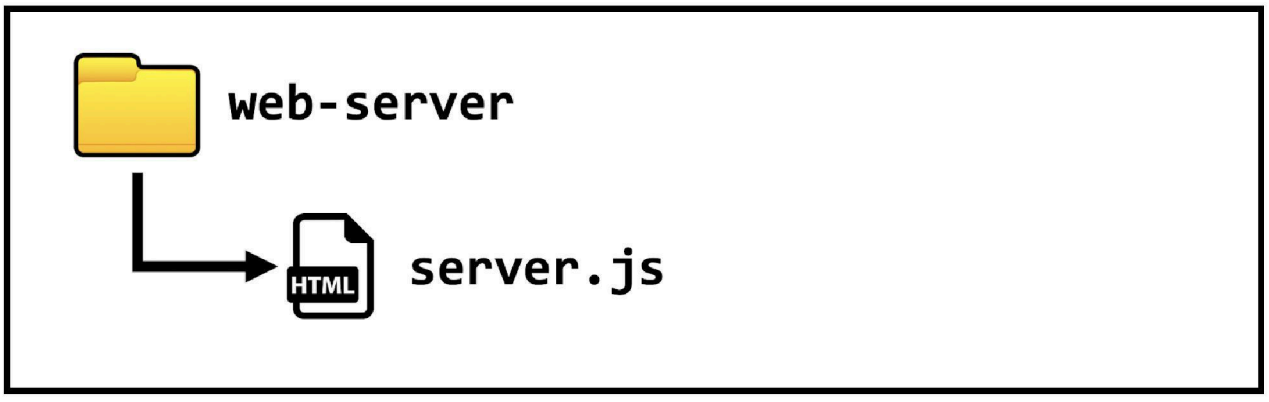
Build a simple web server application in Javascript that runs in Node JS from bash terminal.

Learning Objectives

- NodeJS, NPM
- HTTP, MIME Types, HTTP Error codes
- Routes

Server-side Architecture:

Start this project by making a project folder where all your assets & scripts will be organized.
Create all necessary files and folders as illustrated below.



Concepts

NPM *(Node Package Manager)*

NPM is a package manager for Node.js packages. It installs node dependencies and launches node apps
https://docs.npmjs.com/

Node JS

Node.js is a JavaScript Runtime Environment. Node.js allows you to run JavaScript on the server.
https://nodejs.org/en/

- **Builtin Packages:** Node includes several modules that provide functionality for http servers, file systems, web sockets, crypto, and much more. https://nodejs.org/api/
- **Asynchronous & Event Driven:** All APIs of Node.js library are asynchronous and non-blocking, allowing easy requests to external backend services before sending a response.
- **Terminal Support:** Launch JavaScript applications from your terminal with Node.js

---

Web Server (Backend Service)

A web server is an application that listens for client HTTP requests and sends a response:
https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

- **Routes:** Routing refers to how an application's endpoints (URIs) respond to client requests.
- **MIME Types**: A media type (also known as a Multipurpose Internet Mail Extensions or MIME type) is a standard that indicates the nature and format of a document, file, or assortment of bytes. https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types
- **Status Codes**: HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:
    - Informational responses (100–199),
    - Successful responses (200–299),
    - Redirects responses (300–399),
    - Client error responses(400–499),
    - Server error responses (500–599).

  https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Iteration 1: Web Server deployed with Node

'Approach' → Plan phase

Goal #1: Simple web server that gives a "Hello World" response, deployed with Node

Approach: Use Node and its standard library

Make a web server that listens for HTTP requests from a port and sends a HTTP response.

'Apply' → Do phase

JavaScript Steps

Step 1 (JS): import 'http' module, & initialize server data

Import the http module which contains builtin server functions and initialize variables to define the server's hostname and port number that it'll listen on.

server.js

```
const http = require('http');
const hostname = '127.0.0.1';  //or 'localhost'
const port = 3000;
```

Step 2 (JS): httpHandler (callback function)

A callback function the server invokes when it receives an HTTP request. The server passes the request object and an empty response object to the function. The callback defines the response's attributes.

server.js

```
function httpHandler(request, response){
   response.statusCode = 200;                          //Status Code: 200 - Success
   response.setHeader('Content-Type', 'text/plain');   //Response Content Type: text
   response.end('Hello World\n');                      //End Response, pass data
}
```

Step 3 (JS): serverMessage (callback function)

A callback function the server invokes when it initially launches.

server.js

```
function serverMessage(){
   console.log(`Server running at http://${hostname}:${port}/`)
}
```

Step 4 (JS): launch the server

Create a web server that handles HTTP requests via the callback function. The server listens on the port and hostname. When it launches it triggers the given callback function.

server.js

```
const server = http.createServer(httpHandler)
server.listen(port, hostname, serverMessage)
```

'Assess' → Test phase

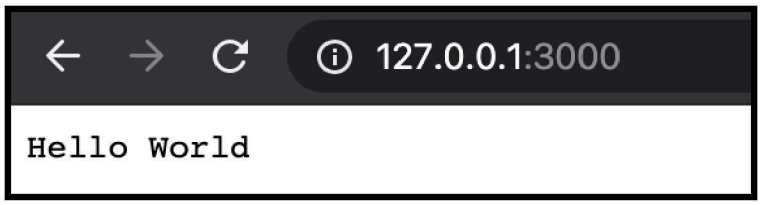
Launch the web server application from the bash terminal with node command:

```
node server.js
```

You should get a display message from the terminal:

```
Server running at http://127.0.0.1:3000/
```

Open browser at the given url address (from above) which should show "hello world" message.



Note: if your hostname = 'localhost' then the url should be: http://localhost:3000

Iteration 2: Web Server deployed with NPM

'Approach' → Plan phase

Goal #2: Create a package.json to deploy web server via NPM

Approach: Use NPM to deploy App

Most backend applications are deployed using NPM via the package,json

---

'Apply' → Do phase

JavaScript Steps

Step 1 (JSON): package.json

Make package.json file that contains the configuration information to use npm as build tool to launch app

package.json

```
{
   "name": "httpServer",
   "version": "0.0.1",
   "description": "Basic HTTP Server",
   "main": "server.js",
   "author": "me"
}
```

'Assess' → Test phase

Launch the web server application from the bash terminal with npm command:

```
npm start
```

You should get a display message from the terminal:

```
Server running at http://127.0.0.1:3000/
```

Open browser at the url address (from above)

Iteration 3: HTML filename from URL path

'Approach' → Plan phase

Goal #3: Parse HTML filename from URL path of the HTTP Request

Approach: Define Response for File not Found (404)

Use url & path modules to parse a reques's url path and send a file not found (404) error to client

---

'Apply' → Do phase

Step 1 (JS): Refactor server.js → data

Import the URL & Path modules

server.js

```
const http = require('http');
const hostname = '127.0.0.1';  //or 'localhost'
const port = 3000;
const url = require('url');
const path = require('path');
```

Step 2 (JS): Refactor server.js → httpHandler (callback function)

Refactor server callback to parse a request's URL object & get the pathname from that url & concatenate the requested path to the current working directory (cwd) of server and display that to console.

server.js

```
function httpHandler(request, response){
   let urlObject = url.parse(request.url);
   let uri = urlObject.pathname;
   let fileName = path.join(process.cwd(), uri);
   console.log('Loading ' + uri);
}
```

'Assess' → Test phase

Launch the web server application from the bash terminal with command:

```
npm start
```

You should get a display message from the terminal:

```
Server running at http://127.0.0.1:3000/
```

Open browser at the url address (from above) & you should see in terminal:

```
Loading /
```

Type a request for subpage in browser such as: **http://127.0.0.1:3000/index.html**

```
Loading /index.html
```

Note: Since the request is never resolved from the backend, there is nothing for the browser to display.

Iteration 4: Respond with 404 errors

'Approach' → Plan phase

Goal #4: Response to all request with 404 errors: file not found

Approach: Define behavior for unsuccessful URL routes in web server

Use file system module to determine if requested file doesn't exist & send 404 error when file not found.

'Apply' → Do phase

Step 1 (JS): Refactor server.js → data

Import the fs modules to access filesystem behaviors

server.js

```
const http = require('http');
const hostname = '127.0.0.1';  //or 'localhost'
const port = 3000;
const url = require('url');
const path = require('path');
const fileSystem = require('fs');
```

Step 2 (JS): Refactor server.js → httpHandler (callback function)

Refactor server callback to try to use the file system to sync with the requested file, if it exists then set stats variable otherwise invoke the helper function: fileNotFound. Note: No files exist on the backend service to serve yet. At this iteration, the client should always throw the exception case for every filename.

server.js

```
function httpHandler(request, response){
   let urlObject = url.parse(request.url);
   let uri = urlObject.pathname;
   let fileName = path.join(process.cwd(), uri);
   console.log('Loading ' + uri);

   let stats;
   try{
       stats = fileSystem.lstatSync(fileName);
   }catch(e){
       fileNotFound(response)
   }
}
```

Step 3 (JS): fileNotFound

Define a function that defines & sends a Response object for when a Requested file (route) does not exist (404).

server.js

```
function fileNotFound(response){
    response.writeHead(404, {'Content-type': 'text/plain'});
    response.write('404 Not Found\n');
    response.end();
    return;
}
```

'Assess' → Test phase

Launch the web server application from the bash terminal with command:

```
npm start
```

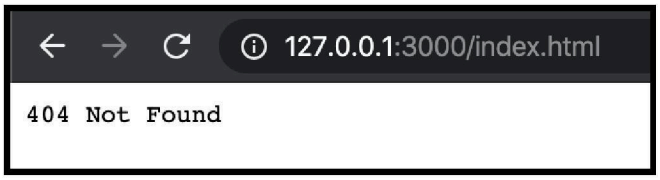
You should get a display message from the terminal:

```
Server running at http://127.0.0.1:3000/
```

Type a request for subpage in browser such as: `http://127.0.0.1:3000/index.html`

```
Loading /index.html
```

GET request from browser (using any URL):



Iteration 5: Respond with 200 success

'Approach' → Plan phase

Goal #5: Response to a successful request with code 200

Approach: Define behavior for successful URL routes in web server

Use file system module to determine if requested file exists & send a 200 Success if file is found.

'Apply' → Do phase

Step 1 (HTML): index.html

Create an index.html file with a simple heading element

index.html

```
<h1>Hello World</h1>
```

Step 1 (JS): server.js → mimeTypes

Define a dictionary that maps file extensions to MIME (content) types

server.js

```
const mimeTypes = {
    ".html": "text/html",
    ".jpeg": "image/jpeg",
    ".jpg":  "image/jpg",
    ".png":  "image/png",
    ".js":   "text/javascript",
    ".css":  "text/css"
};
```

Step 2 (JS): Refactor → httpHandler

Refactor httpHandler to serve file if it exists, invoke helper function serveFile

server.js

```
function httpHandler(request, response){
    let urlObject = url.parse(request.url);
    let uri = urlObject.pathname;
    let fileName = path.join(process.cwd(), uri );
    console.log('Loading ' + uri);

    let stats;
    try{
        stats = fileSystem.lstatSync(fileName);
        if ( stats.isFile() ){
            serveFile(response, fileName)
        }
    }
    catch(e){
        fileNotFound(response)
    }
}
```

Step 3 (JS): serveFile

Define serveFile function that configures Response object, get the associated mime type via file extension and send in response with 200 (good) and content type

server.js

```
function serveFile(response, fileName){
    let fileExtension = path.extname(fileName);   //get file extension to get MIME type
    let mimeType = mimeTypes[fileExtension];
    response.writeHead(200,{'Content-type': mimeType});
    response.write( `200 ${mimeType} Found` );
    response.end();
}
```

'Assess' → Test phase

Launch the web server application from the bash terminal with command:

```
npm start
```

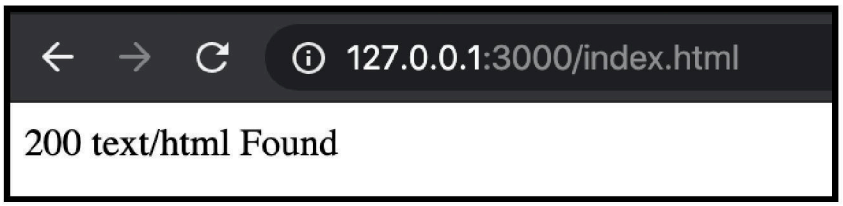
You should get a display message from the terminal:

```
Server running at http://127.0.0.1:3000/
```

Type a request for subpage in browser such as: `http://127.0.0.1:3000/index.html`

```
Loading /index.html
```

GET request from browser for: `http://127.0.0.1:3000/index.html`



Iteration 6: Serve HTML file to Browser

'Approach' → Plan phase

Goal #6: Serve the requested HTML file to Browser

Approach: FileSystem module

Use the File System module to read and send HTML file data to the Browser

---

'Apply' → Do phase

Step 1 (JS): Refactor → serveFile

Refactor serveFile to open file with FileStream object via its filename and read it into the Response object

server.js

```
function serveFile(response, fileName){
    let fileExtension = path.extname(fileName);
    let mimeType = mimeTypes[fileExtension];
    response.writeHead(200,{'Content-type': mimeType});
    let fileStream = fileSystem.createReadStream(fileName);
    fileStream.pipe(response);
}
```

'Assess' → Test phase

Launch the web server application from the bash terminal with command:

```
npm start
```

Type a request for subpage in browser such as: `http://127.0.0.1:3000/index.html`

```
Loading /index.html
```

GET request from browser for: `http://127.0.0.1:3000/index.html`

Iteration 7: Redirect '/' to '/index.html'

'Approach' → Plan phase

Goal #7: Redirect GET '/' Request to return response for '/index.html'

Approach: Use HTTP Code 302: redirect

if the GET request is a directory path in web server, then use HTTP Code 302 to redirect to index.html file
https://developer.mozilla.org/en-US/docs/Web/HTTP/Redirections

---

'Apply' → Do phase

Step 1 (JS): Refactor → httpHandler

If the requested path is a directory, then send the Response object to a helper function: serveIndex

server.js

```
function httpHandler(request, response){
   let urlObject = url.parse(request.url);
   let uri = urlObject.pathname;
   let fileName = path.join(process.cwd(), uri );
   console.log('Loading ' + uri);

   let stats;
   try{
       stats = fs.lstatSync(fileName);
       if ( stats.isFile() ){
           serveFile(response, fileName)
       }
       else if (stats.isDirectory()){
           serveIndex(response);
       }
   }
   catch(e){
       fileNotFound(response)
   }
}
```

Step 2 (JS): serveIndex

Encode Response with 302 code (redirection) to location of 'index.html', then end Response to send back

server.js

```
function serveIndex(response){
   response.writeHead(302, {'Location': 'index.html'})
   response.end();
}
```

'Assess' → Test phase

Launch the web server application from the bash terminal with command:

```
npm start
```

Type a request for subpage in browser such as: `http://127.0.0.1:3000/`

```
Loading /
Loading /index.html
```

GET request from browser for: `http://127.0.0.1:3000/`

Iteration 8: Internal Server Error: 500

'Approach' → Plan phase

Goal #8: Respond with Generic Error if cannot serve File or Directory but it Exists

Approach: HTTP Code 500: Internal Server Error

if the file or directory cannot be served then respond with a 500 error

---

'Apply' → Do phase

Step 1 (JS): Refactor → httpHandler

If the requested path exists, but is not a file or directory then invoke helper function: serveError

server.js

```
function httpHandler(request, response){
    let urlObject = url.parse(request.url);
    let uri = urlObject.pathname;
    let fileName = path.join(process.cwd(), uri );
    console.log('Loading ' + uri);

    let stats;
    try{
        stats = fs.lstatSync(fileName);
        if ( stats.isFile() ){
            serveFile(response, fileName)
        }
        else if (stats.isDirectory()){
            serveIndex(response);
        }
        else{
            serveError();
        }
    }
    catch(e){
        fileNotFound(response)
    }
}
```

Step 2 (JS): serveError

Encode Response with 500 code (Internal Error) then end the Response to send back to the browser.

server.js

```
function serveError(response){
   response.writeHead(500, {'Content-type': 'text/plain'});
   response.write('500 Internal Error\n');
   response.end();
}
```

'Assess' → Test phase

Launch the web server application from the bash terminal with command:

```
npm start
```

Type a request for subpage in browser such as: `http://127.0.0.1:3000/`

```
Loading /
Loading /index.html
```

GET request from browser for: `http://127.0.0.1:3000/`

Note: There is no easy way to serve this error as its an edge case error

Conclusions

Final Comments

In this lab you learned to implement a basic web server in node that responds to HTTP requests and serves HTML files to the web browser. This web server supports 404 File Not Found, 200 Success, 302 Redirection, and 500 Internal Server Error responses. This lab covered: HTTP routes, Mime types, and HTTP codes.

Future Improvements

- Use ExpressJS to deploy a more complex web server with less code

Lab Submission

Compress your project folder into a zip file and submit on Moodle.