Hands-on Lab - First Server with ServerSide Java Script (20 min)

Objective for Exercise:

- Use the terminal to git clone and get Node.JS server code
- Create a web server using Server side Java script
- Run the server
- Access the server from the client and get a response from server

Step 1: Verify Environment and Command-line tools

1. Open a terminal window by using the menu in the editor: Terminal > New Terminal.

2. Verify that `node` CLI is installed.

```
node --version
```

You should see output similar to this, though the versions may be different:

```
v19.9.0
```

3. Change to your project folder.

```
cd /home/project
```

4. Clone the git repository that contains the artifacts needed for this lab, if it doesn't already exist.

```
git clone https://github.com/ibm-developer-skills-network/lkpho-Cloud-applications-with-Node.js-and-React.git
```

5. Change to the directory for this lab.

```
cd lkpho-Cloud-applications-with-Node.js-and-React/CD220Labs/http_server
```

6. List the contents of this directory to see the artifacts for this lab.

```
ls
```

7. Check the content of index.js. This is the server side script we will run in the next section.

```
cat index.js
```

You should see output similar to this.

```
 // Import the HTTP module
const http = require('http');
// Define the request listener function
const requestListener = function (req, res) {
    res.writeHead(200); // Set the status code to 200 (OK)
    res.end('Hello, World!'); // Send the response "Hello, World!"
};
// Define the port number
const port = 8080;
// Create an HTTP server using the request listener function
```

```
const server = http.createServer(requestListener);
// Start the server and listen on the specified port
server.listen(port);
console.log('Server listening on port: ' + port);
```

Explanation:

- `HTTP Module` Import the HTTP module from Node.js.
- `Request Listener Function` Define a request listener function that handles incoming HTTP requests. In this case, the function sets the status code to 200 (OK) and sends the response "Hello, World!".
- `Port Definition` Define the port number as 8080 using const port = 8080;
- `Create HTTP Server` Create an HTTP server using http.createServer(requestListener);, where requestListener is the defined request listener function.
- `Server Start` Start the server and listen on the specified port using server.listen(port);. Also, log a message indicating that the server is listening on the specified port.

Alternatively, you can also view the content of index.js through the file explorer menu. It would appear like this.

Step 2: Use the `node` CLI

1. In order to start the server, we run the index.js file with the node command.

```
node index.js
```

You should see output similar to this.

```
server listening on port: 8080
```

2. To split the terminal, click Split Terminal as shown in image below.

3. In the second terminal window, use the `curl` command to ping the application.

```
curl localhost:8080
```

You should see output similar to this.

```
Hello, World!
```

It should indicate that your app is up and running.

4. To verify the same with browser window, click on the Skills Network button on the left, it will open the "Skills Network Toolbox". Then click the **Other** then **Launch Application**. From there you should be able to enter the port number the server is running on, which is `8080` and launch.

A new browser window will open up as below. (*Note: New browser window may not open up if you browser settings does not allow pop-ups*)

5. To stop the server, go to the main command window and press Ctrl+c to stop the server and stay in that terminal.

Step 3: Use the `node` CLI to run server script which requires another module

1. In the same terminal, check the content of today.js.

```
cat today.js
```

You should see output similar to this.

```
 // Export a function named 'getDate' from the module
module.exports.getDate = function getDate() {
    // Get the current date and time in the timezone "Australia/Brisbane"
    let aestTime = new Date().toLocaleString("en-US", {timeZone: "Australia/Brisbane"});
    return aestTime; // Return the formatted date and time
};
```

Explanation:

- `Export Function` Use module.exports to export a function named getDate from the module.
- `Date Formatting` Inside the getDate function, get the current date and time using new Date().toLocaleString("en-US", {timeZone: "Australia/Brisbane"}). This formats the date and time according to the timezone "Australia/Brisbane".
- `Return Value` Return the formatted date and time using return aestTime;.

Alternatively, you can also view the content of today.js through the file explorer menu. It would appear like this.

We will use this exported module in the server side script.

2. Check the content of index-with-require.js. As you will observe, this script requires the module `today` whose content we saw in the previous step.

```
cat index-with-require.js
```

You should see output similar to this.

```
 // Import the HTTP module
const http = require('http');
// Import the 'today' module
const today = require('./today');
// Define the request listener function
const requestListener = function (req, res) {
    res.writeHead(200); // Set the status code to 200 (OK)
    // Send the response with the current date from the 'today' module
    res.end(`Hello, World! The date today is ${today.getDate()}`);
};
// Define the port number
const port = 8080;
// Create an HTTP server using the request listener function
const server = http.createServer(requestListener);
// Start the server and listen on the specified port
server.listen(port);
console.log('Server listening on port: ' + port);
```

Explanation:

- `HTTP Module` Import the HTTP module from Node.js.

- `Module Import` Import the 'today' module using const today = require('./today');. This assumes that there is a module named ‘today’ exporting a function getDate.
- `Request Listener Function` Define a request listener function that handles incoming HTTP requests. In this case, the function sets the status code to 200 (OK) and sends the response "Hello, World! The date today is {current date}" using the getDate function from the 'today' module.
- `Port Definition` Define the port number as 8080 using const port = 8080;.
- `Create HTTP Server` Create an HTTP server using http.createServer(requestListener);, where requestListener is the defined request listener function.
- `Server Start` Start the server and listen on the specified port using server.listen(port);. Also, log a message indicating that the server is listening on the specified port.

Alternatively, you can also view the content of index-with-require.js through the file explorer menu. It would appear like this.

3. In order to start the server, we run the index-with-require.js file with the node command.

```
node index-with-require.js
```

You should see output similar to this.

```
server listening on port: 8080
```

4. In the second terminal window which you opened earlier, use the `curl` command to ping the application.

```
curl localhost:8080
```

You should see output similar to this.

```
Hello, World! The date today is Wed Oct 14 2020 14:56:42 GMT+1030 (Australian Eastern Standard Time)
```

It should indicate that your app is up and running.

5. To verify the same with browser window, click on the **Skills Network** button on the left, it will open the "Skills Network Toolbox". Then click the **Other** then **Launch Application**. From there you should be able to enter the port number `8080`and launch.

A new browser window will open up which show `Hello World!` along with the date and time in your time zone.

Challenge:

Make changes in index-with-require.js to greet the user **depending** on the time of the day.

▶ Click here for a sample solution

Congratulations! You have completed the lab.

Summary

Now that you have learnt how to run a server you are ready to create your very own Node.JS server.

Author(s)

Lavanya