### Node.JS Lab Summary

In this lab we will

1. Create a basic server using the http module.
2. Add URL parsing with the url module.
3. Serve static files using the fs module.
4. Use the path module for constructing file paths.
5. Add error handling to serve a custom 404 page.

This step-by-step approach hopefully helps you build on your knowledge and progressively incorporate more advanced features.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# First…let’s walk through this together

**Step 1: Initialize a New Node.js Project**

1. Open your terminal and create a new directory for your project using “mkdir” command and the cd into that new folder.
2. Initialize a new Node.js project using “npm init”

**Step 2: Create server.js file in your project**

* Create a server.js file in your project using only a command line. Hint: Use “touch” command.

project-folder/

└── server.js

**Step 3: Work with “http” module to create server that responses to client**

* Open the server.js file and add [the following example code](https://www.w3schools.com/nodejs/shownodejs.asp?filename=demo_http) to your server.js file.
* One you save you can start your server by running the “node server.js” command.
* You can open a browser to the following URL: <http://localhost:3000/>
  + You should see a “Hello World” response written in your browser.
* Without stopping (aka killing) your server, change the res.write to write html instead of just the text reply like so: <h1>Hello World</h1>
* Save your file and refresh your browser. You won’t see the page update. That’s because you need to end the server using CTRL+C and then restart it. Do this and then see if your browser refresh works.

Wouldn’t it be cool if we didn’t have to stop and start the server all the time? Well thankfully we have nodemon package which will auto-restart the server as server code changes are saved.

**Step 4: Install nodemon and then run server with nodemon**

* In your root directory install nodemon as a dev dependency. This means this module only is installed to run on your local dev environment. See if you can figure this out
* One you have nodemon installed restart your server using “nodemon server.js”
* Now see if your browser refreshes after you change your hello world response and save. Nodemon should automatically restart your server when you save. Pretty cool huh?

**Continue on with the following steps to incorporate more core node modules**

### *Step 1: Update the Node.js Server with Core Modules*

1. **Update the Server code** :

const http = require('http');

//create a server object and store in a const called “server”

const server = http.createServer(function (req, res) {

  res.write('Hello World!'); //write a response to the client

  res.end(); //end the response

})

//store your PORT number in a const

const PORT = 8080;

//listen on the server object and log port server is running on

server.listen(PORT, () => {

  console.log(`Server running on port ${PORT}`);

});

1. **Save code and refresh page**

#### Step 2: Add URL Parsing with url Module

**Update the Server Code**: Modify server.js to include the url module:

const http = require('http');

const url = require('url');

const server = http.createServer((req, res) => {

const parsedUrl = url.parse(req.url, true);

const pathname = parsedUrl.pathname;

res.writeHead(200, { 'Content-Type': 'text/plain' });

res.write(`You requested: ${pathname}`);

res.end();

});

const PORT = 5000;

server.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

**Test URL Parsing**: Restart the server and navigate to different URLs

e.g., http://localhost:5000/about) to see the requested pathname.

#### Step 3: Serve Static Files with fs Module

**Create HTML Files**:

Create a public directory and add index.html and about.html files:

mkdir public

touch public/index.html public/about.html

Add basic HTML content to index.html:

<!DOCTYPE html>

<html>

<head>

<title>Home</title>

</head>

<body>

<h1>Welcome to the Home Page</h1>

</body>

</html>

Add basic HTML content to about.html:

<!DOCTYPE html>

<html>

<head>

<title>About</title>

</head>

<body>

<h1>About Us</h1>

</body>

</html>

**Update the Server Code**: Modify server.js to include the fs module and serve static files:

const http = require('http');

const url = require('url');

const fs = require('fs');

const server = http.createServer((req, res) => {

const parsedUrl = url.parse(req.url, true);

const pathname = parsedUrl.pathname;

let filePath = '.' + pathname;

if (filePath === './') {

filePath = './public/index.html';

} else {

filePath = './public' + pathname + '.html';

}

fs.readFile(filePath, (err, data) => {

if (err) {

res.writeHead(404, { 'Content-Type': 'text/html' });

res.write('<h1>404 Not Found</h1>');

res.end();

} else {

res.writeHead(200, { 'Content-Type': 'text/html' });

res.write(data);

res.end();

}

});

});

const PORT = 5000;

server.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

**Test Static File Serving**: Restart the server and navigate to http://localhost:5000 and http://localhost:5000/about to see the HTML content.

#### Step 4: Use path Module for File Paths

**Update the Server Code**: Modify server.js to include the path module and use it for constructing file paths:

const http = require('http');

const url = require('url');

const fs = require('fs');

const path = require('path');

const server = http.createServer((req, res) => {

const parsedUrl = url.parse(req.url, true);

const pathname = parsedUrl.pathname;

let filePath = path.join(\_\_dirname, 'public', pathname === '/' ? 'index.html' : pathname + '.html');

fs.readFile(filePath, (err, data) => {

if (err) {

res.writeHead(404, { 'Content-Type': 'text/html' });

res.write('<h1>404 Not Found</h1>');

res.end();

} else {

res.writeHead(200, { 'Content-Type': 'text/html' });

res.write(data);

res.end();

}

});

});

const PORT = 5000;

server.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

**Test the Server**: Restart the server and navigate to http://localhost:5000 and http://localhost:5000/about to ensure everything works as expected.

#### Step 5: Add Error Handling

**Update the Server Code**: Modify server.js to include more robust error handling:

const http = require('http');

const url = require('url');

const fs = require('fs');

const path = require('path');

const server = http.createServer((req, res) => {

const parsedUrl = url.parse(req.url, true);

const pathname = parsedUrl.pathname;

let filePath = path.join(\_\_dirname, 'public', pathname === '/' ? 'index.html' : pathname + '.html');

fs.readFile(filePath, (err, data) => {

if (err) {

if (err.code === 'ENOENT') {

fs.readFile(path.join(\_\_dirname, 'public', '404.html'), (err, data) => {

res.writeHead(404, { 'Content-Type': 'text/html' });

res.end(data, 'utf8');

});

} else {

res.writeHead(500);

res.end(`Server Error: ${err.code}`);

}

} else {

res.writeHead(200, { 'Content-Type': 'text/html' });

res.end(data, 'utf8');

}

});

});

const PORT = 5000;

server.listen(PORT, () => {

console.log(`Server running on port ${PORT}`);

});

**Create a 404 Page**: Add a 404.html file to the public directory:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Page not Found</title>

</head>

<body>

    <h1>404 - Uh oh!  We didn't find that page</h1>

</body>

</html>

**Test Error Handling**: Restart the server and navigate to a non-existent page (e.g., http://localhost:5000/unknown) to see the 404 error page.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

After you complete this, we now have some experience with base node modules, which we’ll use to connect to backend servers. But thankfully, Express (the “E” in MERN and MEAN) exists to make our code less complicated.

Express.js is a powerful and flexible web application framework for Node.js, making it easier to build web applications and APIs. Here are some core uses I want you to explore:

1. **Setting Up a Basic Server:** Have them create a basic Express server and understand how it simplifies server creation compared to the core **http** module.
2. **Routing**: Explore how to define routes for different HTTP methods (GET, POST, PUT, DELETE) and how to handle route parameters and query strings.
3. **Middleware**: Understand the concept of middleware in Express, including built-in middleware (like **express.json()** and **express.static()**) and creating custom middleware functions.
4. **Serving Static Files:** Use the **express.static** middleware to serve static files such as HTML, CSS, and JavaScript.
5. **Handling Forms and JSON Data:** Learn how to parse and handle form submissions and JSON data using middleware like **express.urlencoded()** and **express.json()**.
6. **Error Handling:** Implement error-handling middleware to manage errors and send appropriate responses.
7. **Building RESTful APIs:** Create RESTful API endpoints and understand how to structure and organize routes and controllers.
8. **Authentication and Authorization:** Explore basic authentication and authorization techniques, such as using sessions, cookies, or JWT (JSON Web Tokens).
9. **Connecting to Databases:** Integrate with databases (like MongoDB, MySQL, or PostgreSQL) to perform CRUD operations.

Before the next class, you’ll be tasked with providing some details on your understanding of these