Batch: A1	**Roll No.: 16010123012**
Experiment / assignment / tutorial No.: 04	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Demonstrate the Use of node.js and Express.js

AIM: To implement the node js Concepts based on the following topics.

Problem Definition:
Consider the basic concepts, which are useful in the creation of an application.

Resources used:
https://nodejs.org/docs/latest/api/

Expected OUTCOME of Experiment:
CO 2: Illustrate the concepts of various front-end, back-end web application development technologies & frameworks using different web development tools.

Books/ Journals/ Websites referred:

1. Shelly Powers Learning Node O' Reilly 2 nd Edition, 2016.

Pre Lab/ Prior Concepts:
Write details about the following content

1. Node.js

Node.js is an open-source, cross-platform JavaScript runtime environment that allows developers to run JavaScript on the server side. It is built on the V8 JavaScript engine developed by Google, which is also used in Chrome. Unlike traditional JavaScript running in the browser (client-side), Node.js enables developers to use JavaScript to build scalable and high-performance server-side applications.

2. Basic Routing in Node.js

In Node.js, routing refers to how a web server manages incoming requests and sends responses to clients. Routing defines how the server handles different HTTP methods (e.g., GET, POST, PUT, DELETE) and URLs.

Routing with Express: Express.js simplifies routing and provides many more features. Here's an example of routing with Express:

```
const express = require('express');
const app = express();
app.get('/', (req, res) => {
res.send('<h1>Welcome to Node.js with Express!</h1>');
```

```
});
app.listen(3000, () => {
console.log('Server running at http://localhost:3000/');
});
```

3. Custom Modules

In Node.js, a module is a reusable block of code that is encapsulated to handle a particular functionality. You can create your own custom modules and import them into your application to keep the code organized. A custom module is created using the module.exports object to expose functions, objects, or variables.

```
module.exports.add = (a, b) => {
return a + b;
};
```

4. Asynchronous Programming

Asynchronous programming allows Node.js to handle multiple operations at the same time without waiting for each operation to complete. It's a key feature of Node.js that enables it to be efficient and scalable, particularly for I/O-bound tasks such as file reading, database queries, and network requests.

Callback Functions: The most basic form of asynchronous programming in Node.js involves callbacks. When an asynchronous function is called, it doesn't return a result immediately. Instead, it takes a callback function, which is called once the operation is completed.

Promises: A more modern way to handle asynchronous operations in Node.js is by using Promises. A promise represents a value that may be available now, or in the future, or never. Promises can be in one of three states: pending, resolved (fulfilled), or rejected.

Async/Await: async and await provide a more readable and synchronous-like syntax for working with promises. An async function always returns a promise, and inside an async function, you can use await to wait for a promise to resolve.

Problem statement: Demonstrate the functionality of each with a simple script

1) Basic Routing:

1. Build First server application using http module
2. Basic routing: Demonstrate it using simple HTML/Json file
3. Demonstrate the callback in node.js

2) **File operation**

- Check Permissions of a File or Directory.
- Checking if a file or a directory exists.
- Determining the line count of a text file.
- Reading a file line by line.
- See the file content through browser.

3) Building your custom modules

-To demonstrate this use some mathematics function to create custom module.

4) Blocking and Non Blocking

Methodology:

1. Installed Node.js runtime and verified installation. Organized the project folder structure for clarity.
2. Built a simple HTTP server using the http module, added routes for HTML and JSON responses, and tested callbacks.
3.File Operations: Implemented functions for permission checks, existence checks, line counting, line-by-line reading, and served file content via a browser-based viewer.
4.Custom Module: Created a math-module.js file exporting reusable arithmetic functions and imported it in Demo.js to test operations.
5.Blocking vs Non-Blocking: Demonstrated synchronous vs. asynchronous.
6. Ran each script independently, verified console outputs.

Implementation Details:

1. sample.html

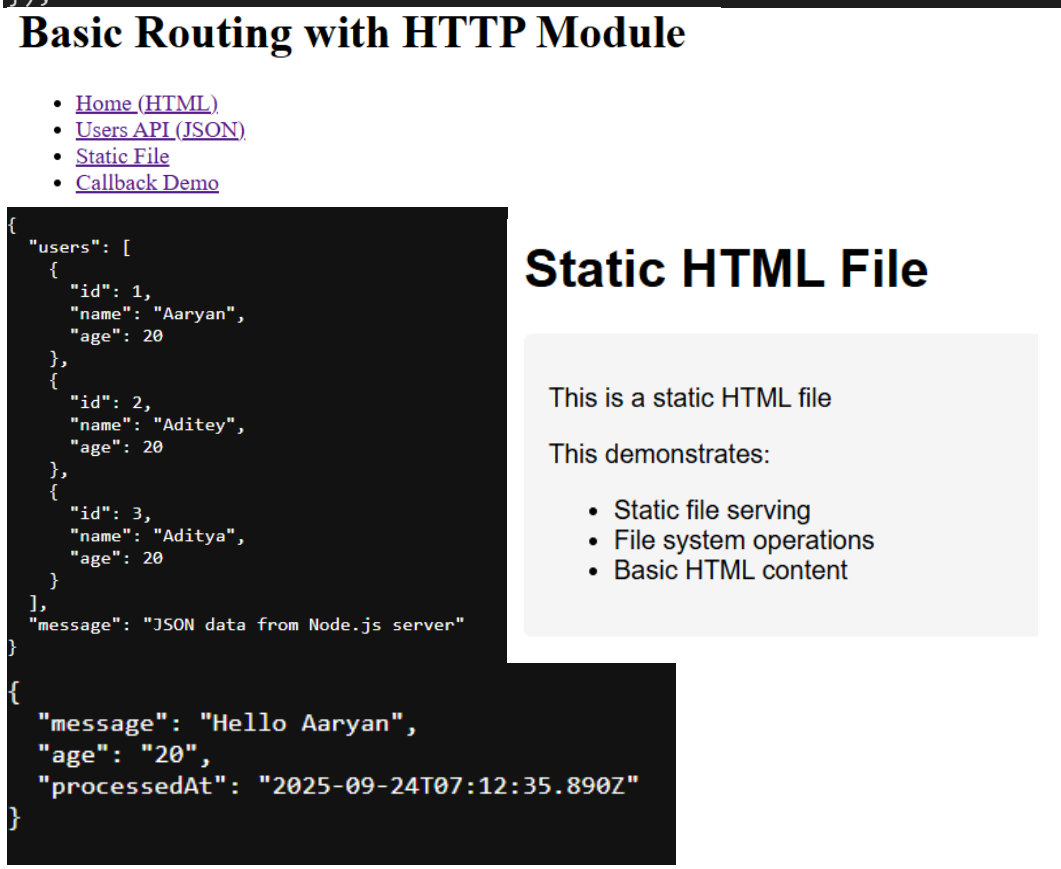
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Static File Demo</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        .content { background: #f5f5f5; padding: 15px; border-radius: 5px; }
    </style>
</head>
<body>
    <h1>Static HTML File</h1>
    <div class="content">
        <p>This is a static HTML file</p>
        <p>This demonstrates:</p>
        <ul>
            <li>Static file serving</li>
            <li>File system operations</li>
            <li>Basic HTML content</li>
        </ul>
    </div>
</body>
</html>
```

Server.js

```
const http = require('http');
const url = require('url');
const fs = require('fs');
const path = require('path');
const PORT = 3001;

function processUserData(userData, callback) {
    console.log('Processing user data with callback...');

    setTimeout(() => {
        const processedData = {
            message: `Hello ${userData.name || 'Anonymous'}`,
            age: userData.age || 'Unknown',
            processedAt: new Date().toISOString()
        };
        callback(null, processedData);
    }, 500);
}

const server = http.createServer((req, res) => {
    const parsedUrl = url.parse(req.url, true);
    const pathname = parsedUrl.pathname;
    const method = req.method;

    console.log(`${method} ${pathname}`);
    if (pathname === '/' && method === 'GET') {
        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.end(`
            <!DOCTYPE html>
            <html>
            <head>
                <title>Basic Routing Demo</title>
            </head>
            <body>
                <h1>Basic Routing with HTTP Module</h1>
                <ul>
                    <li><a href="/">Home (HTML)</a></li>
                    <li><a href="/api/users">Users API (JSON)</a></li>
                    <li><a href="/file">Static File</a></li>
                    <li><a href="/callback?name=Aaryan&age=20">Callback Demo</a></li>
                </ul>
            </body>
```

```
            </html>
        `);
    } else if (pathname === '/api/users' && method === 'GET') {
        res.writeHead(200, { 'Content-Type': 'application/json' });
        res.end(JSON.stringify({
            users: [
                { id: 1, name: 'Aaryan', age: 20 },
                { id: 2, name: 'Aditey', age: 20 },
                { id: 3, name: 'Aditya', age: 20 },
            ],
            message: 'JSON data from Node.js server'
        }));
    } else if (pathname === '/file' && method === 'GET') {
        const filePath = path.join(__dirname, 'sample.html');
        fs.readFile(filePath, (err, data) => {
            if (err) {
                res.writeHead(404, { 'Content-Type': 'text/html' });
                res.end('<h1>File not found</h1>');
            } else {
                res.writeHead(200, { 'Content-Type': 'text/html' });
                res.end(data);
            }
        });
    } else if (pathname === '/callback' && method === 'GET') {
        const userData = {
            name: parsedUrl.query.name,
            age: parsedUrl.query.age
        };

        processUserData(userData, (error, result) => {
            if (error) {
                res.writeHead(500, { 'Content-Type': 'application/json' });
                res.end(JSON.stringify({ error: 'Processing failed' }));
            } else {
                res.writeHead(200, { 'Content-Type': 'application/json' });
                res.end(JSON.stringify(result));
            }
        });
    } else {
        res.writeHead(404, { 'Content-Type': 'text/html' });
        res.end('<h1>404 - Route Not Found</h1>');
    }
});
```

```
server.listen(PORT, () => {
    console.log(`Basic Routing Server running at http://localhost:${PORT}/`);
    console.log('Demonstrates: HTTP module, routing, HTML/JSON responses, callbacks');
});

process.on('SIGINT', () => {
    console.log('\nShutting down routing server...');
    server.close(() => {
        console.log('Routing server closed.');
        process.exit(0);
    });
});
```



```
PS D:\KJSCE\BTech\TY\Sem V\MERN\Code\nodeexp> npm run routing

> nodejs-demonstrations@1.0.0 routing
> node 1-basic-routing/server.js

Basic Routing Server running at http://localhost:3001/
Demonstrates: HTTP module, routing, HTML/JSON responses, callbacks
GET /
GET /
GET /api/users
GET /file
GET /callback
Processing user data with callback...

Shutting down routing server...
Routing server closed.
```

2. file-operations.js

```
const fs = require('fs');
const path = require('path');
const readline = require('readline');
const http = require('http');

console.log('File Operations Demonstration\n');
const testFile = path.join(__dirname, 'demo.txt');
function createDemoFile() {
    const content = `Line 1: Welcome to file operations
Line 2: This demonstrates file handling
Line 3: Using Node.js built-in fs module
Line 4: Check permissions and existence
Line 5: Count lines in this file
Line 6: Read file line by line
Line 7: View through browser
Line 8: End of demo file`;

    try {
        fs.writeFileSync(testFile, content);
        console.log('Demo file created: demo.txt\n');
    } catch (error) {
        console.error('Error creating demo file:', error.message);
    }
}

function checkPermissions(filePath) {
    console.log('1. CHECK PERMISSIONS');
    try {
        fs.accessSync(filePath, fs.constants.F_OK);
        console.log(`File exists: ${filePath}`);

        try {
            fs.accessSync(filePath, fs.constants.R_OK);
```

```
            console.log('Readable: YES');
        } catch {
            console.log('Readable: NO');
        }

        try {
            fs.accessSync(filePath, fs.constants.W_OK);
            console.log('Writable: YES');
        } catch {
            console.log('Writable: NO');
        }

    } catch (error) {
        console.log('File does not exist or no access');
    }
    console.log();
}

function checkFileExists(filePath) {
    console.log('2. CHECK FILE EXISTENCE');
    const exists = fs.existsSync(filePath);
    console.log(`File: ${path.basename(filePath)}`);
    console.log(`Exists: ${exists ? 'YES' : 'NO'}`);
    console.log();
}

function countLines(filePath) {
    console.log('3. COUNT LINES');
    if (!fs.existsSync(filePath)) {
        console.log('File not found');
        return;
    }

    try {
        const content = fs.readFileSync(filePath, 'utf8');
        const lines = content.split('\n');
        console.log(`File: ${path.basename(filePath)}`);
        console.log(`Total lines: ${lines.length}`);
    } catch (error) {
        console.error('Error counting lines:', error.message);
    }
    console.log();
}
```

```
async function readLineByLine(filePath) {
    console.log('4. READ LINE BY LINE');
    if (!fs.existsSync(filePath)) {
        console.log('File not found');
        return;
    }

    try {
        const fileStream = fs.createReadStream(filePath);
        const rl = readline.createInterface({
            input: fileStream,
            crlfDelay: Infinity
        });

        let lineNumber = 0;
        for await (const line of rl) {
            lineNumber++;
            console.log(`Line ${lineNumber}: ${line}`);
        }
        console.log(`Finished reading ${lineNumber} lines`);
    } catch (error) {
        console.error('Error reading line by line:', error.message);
    }
    console.log();
}

function createBrowserViewer() {
    console.log('5. BROWSER FILE VIEWER');
    const PORT = 3002;

    const server = http.createServer((req, res) => {
        if (req.url === '/') {
            res.writeHead(200, { 'Content-Type': 'text/html' });
            res.end(`
                <!DOCTYPE html>
                <html>
                <head>
                    <title>File Viewer</title>
                    <style>
                        body { font-family: Arial, sans-serif; margin: 20px; }
                        .file-content { background: #f5f5f5; padding: 15px; white-
space: pre-wrap; font-family: monospace; }
```

```
                    </style>
                </head>
                <body>
                    <h1>File Content Viewer</h1>
                    <p><a href="/view-file">View demo.txt</a></p>
                </body>
                </html>
            `);
        } else if (req.url === '/view-file') {
            fs.readFile(testFile, 'utf8', (err, data) => {
                if (err) {
                    res.writeHead(404, { 'Content-Type': 'text/html' });
                    res.end('<h1>File not found</h1>');
                    return;
                }

                res.writeHead(200, { 'Content-Type': 'text/html' });
                res.end(`
                    <!DOCTYPE html>
                    <html>
                    <head>
                        <title>File Content</title>
                        <style>
                            body { font-family: Arial, sans-serif; margin: 20px; }
                            .file-content { background: #f5f5f5; padding: 15px; white-
space: pre-wrap; font-family: monospace; border: 1px solid #ddd; }
                        </style>
                    </head>
                    <body>
                        <h1>File: demo.txt</h1>
                        <div class="file-content">${data}</div>
                        <p><a href="/">Back to Home</a></p>
                    </body>
                    </html>
                `);
            });
        } else {
            res.writeHead(404, { 'Content-Type': 'text/html' });
            res.end('<h1>404 - Not Found</h1>');
        }
    });

    server.listen(PORT, () => {
```

```
        console.log(`File viewer server running at http://localhost:${PORT}/`);
        console.log('You can view file content through your browser');
    });

    return server;
}

async function runDemo() {
    createDemoFile();
    checkPermissions(testFile);
    checkFileExists(testFile);
    countLines(testFile);
    await readLineByLine(testFile);

    const server = createBrowserViewer();
    process.on('SIGINT', () => {
        console.log('\nShutting down file operations demo...');
        server.close(() => {
            console.log('Server closed.');
            process.exit(0);
        });
    });
}

module.exports = {
    checkPermissions,
    checkFileExists,
    countLines,
    readLineByLine,
    createBrowserViewer
};

if (require.main === module) {
    runDemo().catch(console.error);
}
```

File Content Viewer

View demo.txt

File: demo.txt

```
Line 1: Welcome to file operations
Line 2: This demonstrates file handling
Line 3: Using Node.js built-in fs module
Line 4: Check permissions and existence
Line 5: Count lines in this file
Line 6: Read file line by line
Line 7: View through browser
Line 8: End of demo file
```

Back to Home

```
PS D:\KJSCE\BTech\TY\Sem V\MERN\Code\nodeexp> npm run file-ops

> nodejs-demonstrations@1.0.0 file-ops
> node 2-file-operations/file-operations.js

File Operations Demonstration

Demo file created: demo.txt

1. CHECK PERMISSIONS
File exists: D:\KJSCE\BTech\TY\Sem V\MERN\Code\nodeexp\2-file-operations\demo.txt
Readable: YES
Writable: YES

2. CHECK FILE EXISTENCE
File: demo.txt
Exists: YES

3. COUNT LINES
File: demo.txt
Total lines: 8

4. READ LINE BY LINE
Line 1: Line 1: Welcome to file operations
Line 2: Line 2: This demonstrates file handling
Line 3: Line 3: Using Node.js built-in fs module
Line 4: Line 4: Check permissions and existence
Line 5: Line 5: Count lines in this file
Line 6: Line 6: Read file line by line
Line 7: Line 7: View through browser
Line 8: Line 8: End of demo file
Finished reading 8 lines

5. BROWSER FILE VIEWER
File viewer server running at http://localhost:3002/
You can view file content through your browser

Shutting down file operations demo...
Server closed.
```

3. math-module.js

```
const arithmetic = {
  add: (a, b) => a + b,
  subtract: (a, b) => a - b,
```

```
  multiply: (a, b) => a * b,
  divide: (a, b) => {
    if (b === 0) {
      throw new Error('Division by zero is not allowed');
    }
    return a / b;
  },
  power: (base, exponent) => Math.pow(base, exponent),
  modulo: (a, b) => a % b
};

module.exports = {
  arithmetic,
  add: arithmetic.add,
  subtract: arithmetic.subtract,
  multiply: arithmetic.multiply,
  divide: arithmetic.divide,
  power: arithmetic.power,
  modulo: arithmetic.modulo
};
```

Demo.js

```
const math = require('./math-module');

console.log('Custom Module Demonstration: Math Operations\n');
console.log(`Addition: 15 + 7 = ${math.add(15, 7)}`);
console.log(`Subtraction: 20 - 8 = ${math.subtract(20, 8)}`);
console.log(`Multiplication: 6 * 4 = ${math.multiply(6, 4)}`);
console.log(`Division: 100 ÷ 5 = ${math.divide(100, 5)}`);
console.log(`Power: 3⁴ = ${math.power(3, 4)}`);
console.log(`Modulo: 17 % 3 = ${math.modulo(17, 3)}`);

console.log('\nError Handling:');
try {
    math.divide(10, 0);
} catch (error) {
    console.log(`Division by zero: ${error.message}`);
}

console.log('\nCustom module demonstration completed!');
```

```
PS D:\KJSCE\BTech\TY\Sem V\MERN\Code\nodeexp> npm run custom-module

> nodejs-demonstrations@1.0.0 custom-module
> node 3-custom-modules/demo.js

Custom Module Demonstration: Math Operations

Addition: 15 + 7 = 22
Subtraction: 20 - 8 = 12
Multiplication: 6 * 4 = 24
Division: 100 ÷ 5 = 20
Power: 3⁴ = 81
Modulo: 17 % 3 = 2

Error Handling:
Division by zero: Division by zero is not allowed

Custom module demonstration completed!
```

4. demo.js

```
const fs = require('fs');
const path = require('path');

console.log('Blocking vs Non-blocking Operations Demo\n');

const demoFile = path.join(__dirname, 'test.txt');
const content = 'This is test content for blocking vs non-blocking
demo.\n'.repeat(100);

try {
    fs.writeFileSync(demoFile, content);
    console.log('Demo file created\n');
} catch (error) {
    console.error('Error creating file:', error.message);
}

console.log('1. BLOCKING OPERATIONS');
console.time('Blocking read');
try {
    const data = fs.readFileSync(demoFile, 'utf8');
    console.log(`File read synchronously - Size: ${data.length} characters`);
} catch (error) {
    console.error('Blocking read error:', error.message);
}
```

```
console.timeEnd('Blocking read');
console.log('Main thread was blocked during file read\n');

console.log('2. NON-BLOCKING OPERATIONS');
console.time('Non-blocking read');
fs.readFile(demoFile, 'utf8', (err, data) => {
    if (err) {
        console.error('Non-blocking read error:', err.message);
        return;
    }
    console.log(`File read asynchronously - Size: ${data.length} characters`);
    console.timeEnd('Non-blocking read');

    cleanup();
});

function cleanup() {
    try {
        fs.unlinkSync(demoFile);
        console.log('\nDemo file cleaned up');
    } catch (error) {
        console.error('Cleanup error:', error.message);
    }
}

module.exports = {
    demonstrateBlocking: () => {
        const data = fs.readFileSync(demoFile, 'utf8');
        return data.length;
    },
    demonstrateNonBlocking: (callback) => {
        fs.readFile(demoFile, 'utf8', (err, data) => {
            if (err) return callback(err);
            callback(null, data.length);
        });
    }
};
```

```
PS D:\KJSCE\BTech\TY\Sem V\MERN\Code\nodeexp> npm run blocking

> nodejs-demonstrations@1.0.0 blocking
> node 4-blocking-nonblocking/demo.js

Blocking vs Non-blocking Operations Demo

Demo file created

1. BLOCKING OPERATIONS
File read synchronously - Size: 5600 characters
Blocking read: 0.266ms
Main thread was blocked during file read

2. NON-BLOCKING OPERATIONS
File read asynchronously - Size: 5600 characters
Non-blocking read: 1.945ms

Demo file cleaned up
```

Steps for execution:

1. Create all the necessary files
2. Install Node.js and verify with node -v, Navigate to the project folder in a terminal.
3. Run Basic Routing Server
4. Execute File Operations
5. Test Custom Module
6. Demonstrate Blocking vs Non-Blocking

Conclusion:

I have successfully completed the experiment by creating and running Node.js applications that demonstrate basic routing, file operations, custom modules, and the difference between blocking and non-blocking I/O. This provided practical experience with server-side JavaScript, asynchronous programming, and modular code design using Node.js and Express.js.