# EXPERIMENT – 3

## Aim:

Implement programs using NodeJS

## Description:

Node.js is an open-source, cross-platform runtime environment that allows developers to execute JavaScript code outside of a web browser. Built on the V8 JavaScript engine, which is the same engine powering Google Chrome, Node.js is designed for building fast and scalable network applications.

Node.js is particularly well-suited for:

- **Non-blocking, asynchronous programming:** Its event-driven architecture ensures high performance and efficient use of system resources, making it ideal for handling multiple simultaneous connections.
- **Server-side scripting:** With Node.js, JavaScript can be used on both the client and server side, enabling full-stack development with a single language.
- **Real-time applications:** Examples include chat applications, live updates, and online gaming.

Key features of Node.js:

1. **Single-threaded with event looping:** Node.js handles multiple connections simultaneously through its non-blocking I/O operations, ensuring high throughput.
2. **NPM (Node Package Manager):** A robust package manager for managing dependencies and libraries, allowing developers to reuse code easily.
3. **Cross-platform compatibility:** Node.js can run on multiple operating systems, including Windows, macOS, and Linux.

## Programs:

**1. Write a JS program to implement File operations using File stream module in Node.JS**

This program demonstrates basic file operations in Node.js using the fs (File System) and path modules. The following operations are performed on a file named example.txt:

1. **Write a File:** Creates and writes text to the file fs.writeFile(filePath, data, 'utf8', callback);
2. **Read a File:** Reads and displays the file content fs.readFile(filePath, 'utf8', callback);
3. **Append to a File:** Adds additional text to the file without overwriting existing content fs.appendFile(filePath, data, callback);
4. **Delete a File:** Removes the file from the system fs.unlink(filePath, callback);

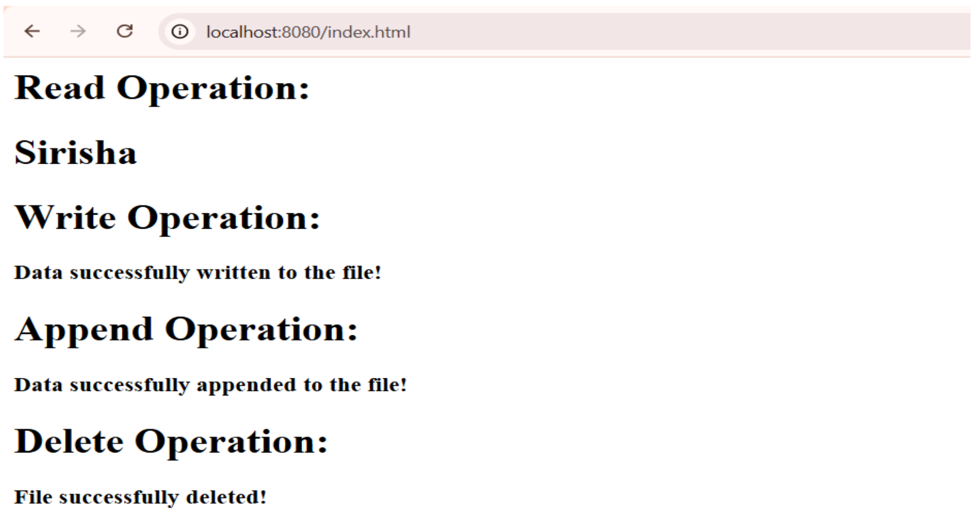
**Program:**

**//server.js**

```
let http = require('http');
let fs = require('fs');
let url = require('url');

http.createServer(function (req, res) {
  var q = url.parse(req.url, true);
  var filename = "." + q.pathname;
  // Read operation
  fs.readFile(filename, function (err, data) {
    res.writeHead(200, { 'Content-Type': 'text/html' });
    if (err) {
      res.write("<h1>404 Not Found</h1>");
    } else {
      res.write('<h1>Read Operation:</h1>');
      res.write(data);
    }
    // Write operation
    fs.writeFile(filename, "<h3>I'm writing in this file..!</h3>", function (err) {
      if (err) {
        res.write('<h1>Write operation failed!</h1>');
        return res.end();
      }
      res.write('<h1>Write Operation:</h1>');
      res.write("<h3>Data successfully written to the file!</h3>");
      // Append operation
```

```
        fs.appendFile(filename, "<h3>Appended text: More content added.</h3>", function (err) {
          if (err) {
            res.write('<h1>Append operation failed!</h1>');
            return res.end();
          }
          res.write('<h1>Append Operation:</h1>');
          res.write("<h3>Data successfully appended to the file!</h3>");
          // Delete operation
          fs.unlink(filename, function (err) {
            if (err) {
              res.write('<h1>Delete operation failed!</h1>');
              return res.end();
            }
            res.write('<h1>Delete Operation:</h1>');
            res.write("<h3>File successfully deleted!</h3>");
            res.end();
          });
        });
      });
    });
}).listen(8080, () => {
  console.log("Server is running on http://localhost:8080");
});
```

**Output:**



```
localhost:8080/index.html

Read Operation:
Sirisha
Write Operation:
Data successfully written to the file!
Append Operation:
Data successfully appended to the file!
Delete Operation:
File successfully deleted!
```

**Fig 1: File operations output**

**2. Write a JS program to implement basic calculator operations on Node.js by using user defined Calc module.**

This program implements a basic calculator in Node.js that performs operations like addition, subtraction, multiplication, and division.

**Modules Used:**

1. **calc.js (User-defined module):**
   - Contains methods for performing arithmetic operations: add, subtract, multiply, and divide.
   - Handles division by zero to avoid runtime errors.
2. **readline (Built-in module):**
   - Provides an interface for reading input from the user via the console.
   - Used to take numbers and the desired operation from the user interactively.

**Program:**

**// calc.js - Calculator module**

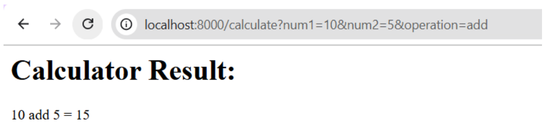
```
module.exports.add = function(a, b) {
   return a + b;
  };
module.exports.subtract = function(a, b) {
   return a - b;
};
module.exports.multiply = function(a, b) {
   return a * b;
};
module.exports.divide = function(a, b) {
if (b === 0) {
   throw new Error("Cannot divide by zero");
}
   return a / b;
};
```

**//app.js**

```
const calc = require('./calci');
http.createServer(function (req, res) {
   var q = url.parse(req.url, true);
   var pathname = q.pathname;
   var query = q.query;
   if (pathname === '/calculate') {
     let num1 = parseFloat(query.num1);
```

```
    let num2 = parseFloat(query.num2);
    let operation = query.operation;
    let result;
    try {
      if (operation === 'add') {
        result = calc.add(num1, num2);
      } else if (operation === 'subtract') {
        result = calc.subtract(num1, num2);
      } else if (operation === 'multiply') {
        result = calc.multiply(num1, num2);
      } else if (operation === 'divide') {
        result = calc.divide(num1, num2);
      } else {
        throw new Error("Invalid operation");
      }
      res.writeHead(200, { 'Content-Type': 'text/html' });
      res.write(`<h1>Calculator Result:</h1>`);
      res.write(`<p>${num1} ${operation} ${num2} = ${result}</p>`);
    } catch (error) {
      res.writeHead(400, { 'Content-Type': 'text/html' });
      res.write("<h1>Error:</h1>");
      res.write(`<p>${error.message}</p>`);
    }
    return res.end();
  }
  res.writeHead(404, { 'Content-Type': 'text/html' });
  res.write("<h1>Page Not Found</h1>");
  return res.end();
}).listen(8000, () => {
  console.log("Server is running on http://localhost:8000");
});
```

**Output:**



localhost:8000/calculate?num1=10&num2=5&operation=add

**Calculator Result:**

10 add 5 = 15

**Fig 2: Calculator output**

## 3. Register and Publish user package in NPM global registry

This experiment demonstrates the process of creating and publishing a npm package. The package.json file contains key metadata for the package:

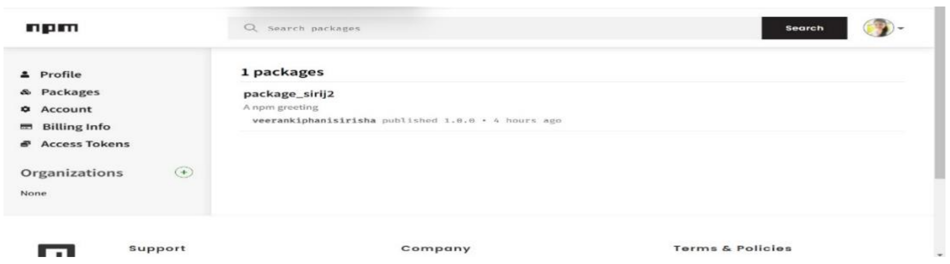
- **Package Name:** The unique identifier for the npm package.
- **Version:** Specifies the current version of the package using semantic versioning.
- **Description:** A brief summary of the package's purpose or functionality.

**//package.json**

```
{
  "name": "package_sirij2",
  "version": "1.0.0",
  "description": "a npm greeting",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "(git+https://github.com/Phanisirisha-46/package_j2)"
  },
  "author": "siri",
  "license": "ISC"
}
```

```
//index.js
function npmHello() {
   console.log('Hello from npm!');
}
module.exports = npmHello;
```

**Output:**



1 packages

package_sirij2

A npm greeting

veerankiphanisirisha published 1.0.0 • 4 hours ago

**Fig 3: npm package published**

4. **Write a JS program to establish client server communication using HTTP module in NODE.JS**

   This program demonstrates communication between a client (web browser) and a server (Node.js) using the HTTP module in Node.js.

   Server-Side (Node.js):

   - The server is created using the http module.
   - It listens on port 3000 and responds to GET requests at the root URL (/).
   - The server sends a JSON response with a message: "Hello from the Node.js server!" when accessed.
   - CORS headers are set to allow cross-origin requests from any domain.

   Client-Side (HTML & JavaScript):

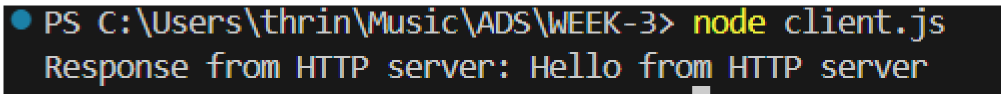
   - The frontend is a simple HTML page with a button that, when clicked, sends a GET request to the Node.js server.
   - The fetch API is used to make the request to http://localhost:3000/.
   - On success, the response from the server is displayed on the webpage.
   - If there's an error, it shows the error message.

**Program:**

**//server.js:**

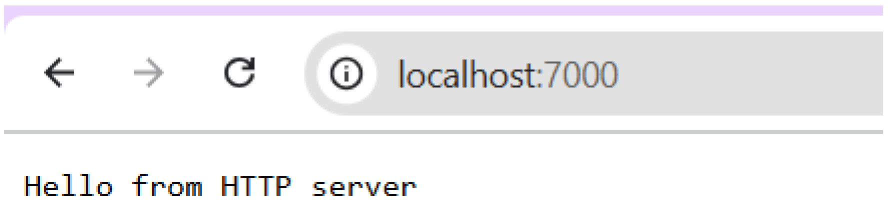
```
const http = require('http');

const server = http.createServer((req, res) => {
  if (req.method === 'GET') {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello from HTTP server');
  } else {
    res.writeHead(405, { 'Content-Type': 'text/plain' });
    res.end('Method Not Allowed');
  }
});

server.listen(7000, () => {
  console.log('HTTP server is running on http://localhost:7000');
});

//client.js:
const http = require('http');
const options = {
  hostname: 'localhost',
  port: 8080,
  path: '/',
  method: 'GET',
};
const req = http.request(options, (res) => {
  let data = '';
  res.on('data', (chunk) => {
    data += chunk;
  });
  res.on('end', () => {
    console.log('Response from HTTP server:', data);
  });
});
req.on('error', (err) => {
  console.error(`Error: ${err.message}`);
});
req.end();
```

**Output:**



```
PS C:\Users\thrin\Music\ADS\WEEK-3> node client.js
Response from HTTP server: Hello from HTTP server
```

**Fig 4: Terminal output of server running**



```
localhost:7000

Hello from HTTP server
```

**Fig 5: Node.js server running on localhost:7000**

## Result:

Programs are implemented using Nodejs successfully.