Experiment No.6
Node.Js: Installation and Configuration, Callbacks, Event loops, Creating express
Name: Shruti Gauchandra
Roll Number: 18
Date of Performance: 01/08/25
Date of Submission: 12/08/25
Marks:
Sign:

Aim: To implement file system operations and path handling in Node.js, and to design a simple REST API using Express framework with middleware functions.

Objective:
To understand the usage of File System (fs) module for reading, writing, and updating files
To learn how to work with path module for handling file and directory paths
To develop RESTful API endpoints using Express framework
To apply middleware in Express for logging and request handling

Requirement:
Node.js installed (version 18 or above recommended)
Express framework (installed via npm)
Text editor such as VS Code
Web browser or Postman for API testing
Theory:
Node.js provides inbuilt modules and frameworks for backend development.
File System Module (fs):
This module allows reading, writing, updating, and deleting files. Example functions include fs.readFileSync and fs.writeFileSync.
Path Module (path):
This module helps in working with file and directory paths. Example usage is path.join(dirname, 'file.txt').
Express Framework:
Express is a lightweight Node.js framework to build web servers and REST APIs. It provides routing, middleware, and response handling.
Middleware in Express:
Middleware functions are executed between request and response. They are used for logging, authentication, validation, and other purposes.

Procedure:

Step 1: Install Node.js and initialize the project using the following commands

```
mkdir node-experiment
cd node-experiment
npm init -y
npm install express
```

Step 2: Create server.js file and include required modules

```
const fs = require('fs');
const path = require('path');
const express = require('express');
const app = express();
const PORT = 3000;
```

Step 3: Perform File System operations

```
fs.writeFileSync('sample.txt', 'Hello, Node.js FS Module!');
const data = fs.readFileSync('sample.txt', 'utf8');
console.log("File Content:", data);
```

Step 4: Use Path Module

```
const filePath = path.join(__dirname, 'sample.txt');
console.log("Absolute File Path:", filePath);
```

Step 5: Create Express REST API with Middleware

```
app.use((req, res, next) => {
   console.log(`${req.method} ${req.url}`);
   next();
});

app.get('/', (req, res) => res.send('Welcome to Node.js REST API'));
app.get('/data', (req, res) => res.json({ message: "Hello World" }));

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

Step 6: Run the server using the command

```
node server.js
```

Step 7: Test endpoints using browser or Postman

http://localhost:3000/ → Displays welcome message

http://localhost:3000/data → Returns JSON response

Output:

File is created and read successfully using File System module

Path module displayed absolute file path

Express server runs successfully with REST API endpoints

Middleware logs request details in console

Conclusion:

Node.js provides powerful modules like fs and path for file handling. Using Express, developers can easily create REST APIs and enhance functionality with middleware. This experiment demonstrates integration of core Node.js modules with Express framework for backend development. The experiment successfully demonstrates the integration of Node.js core modules with the Express framework for backend development. Using modules like fs and path, efficient file handling and path management were achieved. The use of Express simplified the creation of RESTful APIs and enhanced functionality through middleware support. Overall, this experiment highlights how Node.js and Express together provide a powerful, flexible, and efficient environment for building modern server-side applications.