Aim:

To understand and implement different modules in Node.js, specifically the Networking, File System, and Web modules in a single program.

Theory:

Node.js has a modular architecture, where different functionalities are encapsulated in modules. These modules can be built-in, third-party, or user-defined. Three important built-in modules in Node.js are:

1. **Networking Module (`net`)** - Used for creating network applications such as TCP or UDP servers and clients.
2. **File System Module (`fs`)** - Used to handle file operations such as reading, writing, updating, and deleting files.
3. **Web Module (`http`)** - Used to create web servers that handle HTTP requests and responses.

Additionally, we used the **Mammoth** module to extract readable text from a `.docx` file, which helped display its contents on a web server.

Requirements:

- Node.js installed on the system
- Code editor (e.g., VS Code)
- Terminal or command prompt
- `mammoth` module installed using `npm install mammoth`

Implementation:

Original Code:

Initially, we wrote a program that included the `net`, `fs`, and `http` modules to demonstrate their usage:

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

// 1. Networking Module - Create a TCP server
const tcpServer = net.createServer((socket) => {
    socket.write('Hello from TCP Server!\n');
    socket.end();
});

tcpServer.listen(8080, () => {
    console.log('TCP Server running on port 8080');
});

// 2. File System Module - Read a file
const filePath = 'H:\\GVAIET\\Sem 6\\TE IoT\\WEBX.0\\MODULES\\Practical
3.docx';
let fileContent = fs.readFileSync(filePath, 'utf8');
console.log('File Content:', fileContent);
```

```
// 3. Web Module - Create an HTTP Server
const webServer = http.createServer((req, res) => {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello, World! This is a Node.js web server.\nFile Content: ' +
fileContent);
});

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

Issues Faced and Solutions:

1. **Unreadable `.docx` Content:**
   - The file was binary and could not be read as plain text.
   - **Solution:** We integrated the `mammoth` module to extract readable text.
2. **`mammoth` Module Not Found Error:**
   - This occurred because the module was not installed.
   - **Solution:** Installed it using `npm install mammoth`.
3. **Port 8080 Already in Use (`EADDRINUSE` Error):**
   - This happened when another process was using the same port.
   - **Solution:** We identified and killed the process using `netstat -ano | findstr :8080` followed by `taskkill /PID <PID> /F`.
   - Alternatively, we changed the port.

Updated Code with `mammoth` Integration:

```
const net = require('net');
const fs = require('fs');
const http = require('http');
const mammoth = require('mammoth');

// Define the file path
const filePath = 'H:\\GVAIET\\Sem 6\\TE IoT\\WEBX.0\\MODULES\\Practical
3.docx';

// 1. Networking Module - Create a TCP server
const tcpServer = net.createServer((socket) => {
    socket.write('Hello from TCP Server!\n');
    socket.end();
});

tcpServer.listen(8081, () => {
    console.log('TCP Server running on port 8081');
});

// 2. File System Module - Read and extract text from a .docx file
let fileContent = 'File content could not be extracted.';
fs.readFile(filePath, (err, data) => {
    if (err) {
        console.error('Error reading the file:', err.message);
    } else {
        mammoth.extractRawText({ buffer: data })
            .then(result => {
                fileContent = result.value || 'No readable text found in
document.';
                console.log('Extracted Text:', fileContent);
            })
            .catch(err => console.error('Error extracting text:', err));
```

```
    }
});

// 3. Web Module - Create an HTTP Server to display extracted text
const webServer = http.createServer((req, res) => {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello, World! This is a Node.js web server.\nExtracted File
Content: ' + fileContent);
});

webServer.listen(3001, () => {
    console.log('Web Server running on http://localhost:3001');
});
```

Running the Program:

Run the program with:

```
npm install mammoth
node app.js
```

- The TCP server will now run on port 8081.
- The `.docx` file `Practical 3.docx` will be read and converted to plain text.
- The web server will run on `http://localhost:3001` and display the extracted text.

Output:

- The networking module creates a TCP server that responds with a message.
- The file system module reads and extracts text from the `.docx` file using `mammoth`.
- The web module creates an HTTP server displaying the extracted text from the file.

Conclusion:

In this practical, we explored the `net`, `fs`, `http`, and `mammoth` modules in Node.js within a single program. The `mammoth` module was used to extract readable text from a `.docx` file, allowing us to display it on a web server. We also addressed several errors encountered during implementation, such as the unreadable `.docx` issue, module installation errors, and port conflicts. This demonstrates the flexibility of Node.js in handling networking, file operations, and web development.