ASSIGNMENT 1

Source Code:

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

const bodyParser = require('body-parser');


const devicesFilePath = path.join(__dirname +"/"+ "public" + "/"+
"devices.json");

let devices = [];

// Read the file using a relative path
try {
  const data = fs.readFileSync(devicesFilePath);
  const devices = JSON.parse(data);
} catch (err) {
  console.error('Error reading devices file:', err);
}


let existing_devices = loadDevices();
// Function to load existing devices from devices.json
function loadDevices() {
try {
  const data = fs.readFileSync(devicesFilePath);
  return JSON.parse(data);
} catch (err) {
  console.error('Error reading devices file:', err);
  return [];
}
}

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));


app.use(express.json());

// Serve static files from the 'public' directory
app.use(express.static('public'));
```

```
// Get method to display the Webpage index.html

app.get('/index.html', function (req, res) {
    res.sendFile(__dirname + "/" + "index.html");
})


// Function to log activities with timestamps to logs.txt file
function logData(activity, details) {

  const timestamp = new Date().toISOString();
  const logEntry = `[${timestamp}] ${activity}: ${details}\n`;

  const logsFilePath = path.join(__dirname +"/"+ "public" + "/", 'logs.txt');

  fs.appendFile(logsFilePath, logEntry, (err) => {
    if (err) {
      console.error('Error writing to logs file:', err);
    }
  });
}

module.exports = logData;


// Endpoint to register new devices
app.post('/register', (req, res) => {

    var response = {
        deviceId : req.body.deviceId,
        deviceType : req.body.deviceType
    }

    // Storing deviceId and deviceType in below variables

    let deviceId = req.body.deviceId;
    let deviceType = req.body.deviceType;

  // Check for duplicate device ID
  if (devices.find(device => device.deviceId === deviceId)) {
    return res.status(409).send('Device ID already exists');
  }

  // Add new device
  devices.push({ deviceId, deviceType });

  // Write updated devices array to devices.json file
  fs.writeFile(devicesFilePath, JSON.stringify(devices, null, 2), (err) => {
    if (err) {
      console.error('Error writing to devices file:', err);
      return res.status(500).json({ error: 'Failed to register device' });
    }

    res.setHeader('Content-Type', 'text/html');
    res.status(201).send('<h1>'+"Device registered successfully"+'</h1>');
  });

  // Logging the data to the logs.txt for the device registering task

  logData('Registration', JSON.stringify(req.body));

});

app.get('/show', (req, res) => {
    try {
      // Read devices from devices.json
      const data = fs.readFileSync(devicesFilePath);
      const devices = JSON.parse(data);
      if (devices.length === 0) {
        return res.status(404).json({ error: 'No devices registered' });
      }
      res.json(devices);
      logData('Show devices', res.json(devices));
    } catch (err) {
      console.error('Error reading devices file:', err);
      res.status(500).json({ error: 'Failed to retrieve devices' });
    }
  });


// Endpoint to receive device data
app.post('/data', (req, res) => {
  const { deviceId, data } = req.body;


  // Checking of the device ID exist so that we can send a data to it
  if (devices.find(device => device.deviceId === deviceId))
  {
    // Ensure deviceId and data are present in the request body
    if (!deviceId || !data) {
      return res.status(400).json({ error: 'Both deviceId and data are
required' });
    }

    // Log the received data with a timestamp
    const timestamp = new Date().toISOString();
    console.log(`Received data from device ${deviceId} at ${timestamp}:`,
data);

    logData('Received Data', JSON.stringify(req.body));

    res.status(200).json({ message: 'Data received successfully' });

  }
  else{
    res.status(400).json({message:' Device ID doesnt exist'})
  }
});


app.post('/command', (req, res) => {
  const { deviceId, command } = req.body;

  // Checking of the device ID exist so that we can send a command to it
  if (devices.find(device => device.deviceId === deviceId))
  {

  // Validate the presence of deviceId and command in the request body
  if (!deviceId || !command) {
    return res.status(400).json({ error: 'Both deviceId and command are
required' });
  }

  logData('Sent Command', JSON.stringify(req.body));

  // Log the command with a timestamp
  const timestamp = new Date().toISOString();
  console.log(`Command sent to device ${deviceId} at ${timestamp}:`, command);

  // Respond with a confirmation message
  res.status(200).json({ message: 'Command sent successfully' });
}
else{
    res.status(400).json({message:' Device ID doesnt exist'})
}
});

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

   // Check if existing devices array is not empty
   if (existing_devices.length > 0) {
    console.log('Existing devices are loaded from devices.json:');
    console.log(existing_devices);
  } else {
    console.log('No existing devices found in devices.json.');
}
});
```

API documentation

~~~
Below is the API documentation for  Node.js application:

---

# IoT Device Management System API Documentation

## Register a New Device

Registers a new device with a unique `deviceId` and `deviceType`.

- **URL:** `/register`
- **Method:** `POST`
- **Request Body:**
  ```json
  {
    "deviceId": "string",
    "deviceType": "string"
  }
  ```
- **Success Response:**
  - **Code:** 201 CREATED
    **Content:** `{ "message": "Device registered successfully" }`
- **Error Response:**
  - **Code:** 409 CONFLICT
    **Content:** `{ "error": "Device ID already exists" }`
  - **Code:** 400 BAD REQUEST
    **Content:** `{ "error": "Device ID and Device Type are required" }`
  - **Code:** 500 INTERNAL SERVER ERROR
    **Content:** `{ "error": "Failed to register device" }`

## Display All Registered Devices

Displays all registered devices.

- **URL:** `/show`
- **Method:** `GET`
- **Success Response:**
  - **Code:** 200 OK
    **Content:**
    ```json
~~~

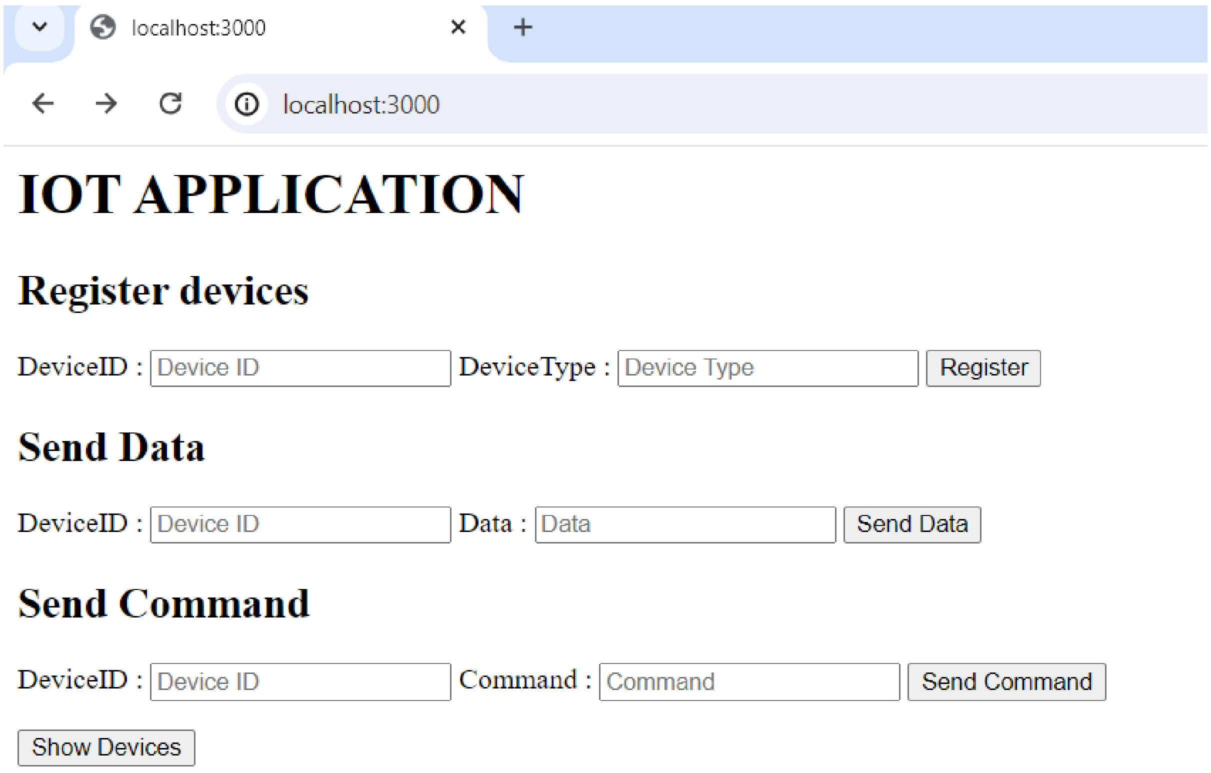
~~~
    [
      { "deviceId": "string", "deviceType": "string" },
      ...
    ]
    ```
- **Error Response:**
  - **Code:** 500 INTERNAL SERVER ERROR
    **Content:** `{ "error": "Failed to retrieve devices" }`

## Receive Data from Device

Receives data from a device.

- **URL:** `/data`
- **Method:** `POST`
- **Request Body:**
  ```json
  {
    "deviceId": "string",
    "data": "string"
  }
  ```
- **Success Response:**
  - **Code:** 200 OK
    **Content:** `{ "message": "Data received successfully" }`
- **Error Response:**
  - **Code:** 400 BAD REQUEST
    **Content:** `{ "error": "Both deviceId and data are required" }`

## Send Command to Device

Sends a command to a device.

- **URL:** `/command`
- **Method:** `POST`
- **Request Body:**
  ```json
  {
    "deviceId": "string",
    "command": "string"
  }
  ```
- **Success Response:**
  - **Code:** 200 OK
    **Content:** `{ "message": "Command sent successfully" }`
- **Error Response:**
  - **Code:** 400 BAD REQUEST
    **Content:** `{ "error": "Both deviceId and command are required" }`
~~~

```
---

This API documentation provides detailed information about the endpoints,
their request bodies, and possible responses, including error handling. It can
help users understand how to interact with your IoT Device Management System.
```

Readme file

```
# IoT Device Management System

This Node.js application allows you to manage IoT devices by providing
endpoints to register new devices, display registered devices, receive data
from devices, and send commands to devices.

## Setup

1. Clone the repository:
git clone <repository-url>
2. Install dependencies:
npm install
npm install express (TO install express framework)

3. Start the server:
npm app


4. Access the application in your browser at `http://localhost:3000`.

## Usage

- Register a new device:
- Endpoint: POST `/register`
- Request Body:


- Display all registered devices:
- Endpoint: GET `/show`

- Receive data from a device:
- Endpoint: POST `/data`
- Request Body:


- Send a command to a device:
- Endpoint: POST `/command`
- Request Body:
```

Output Screenshots



IOT APPLICATION

Register devices

DeviceID : Device ID DeviceType : Device Type Register

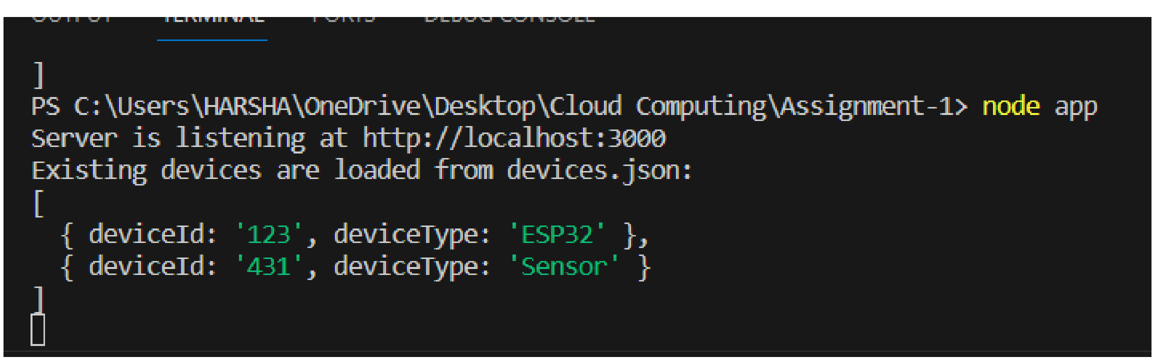
Send Data

DeviceID : Device ID Data : Data Send Data

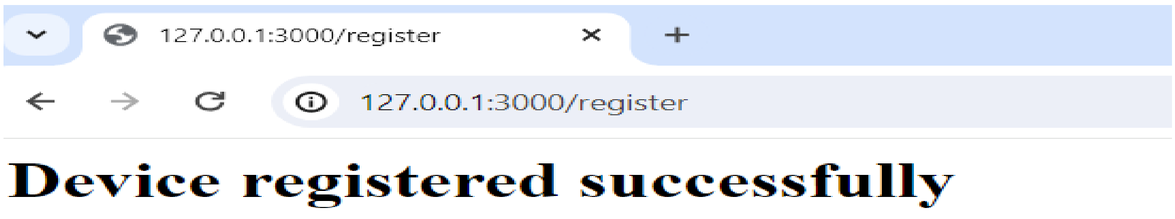
Send Command

DeviceID : Device ID Command : Command Send Command

Show Devices

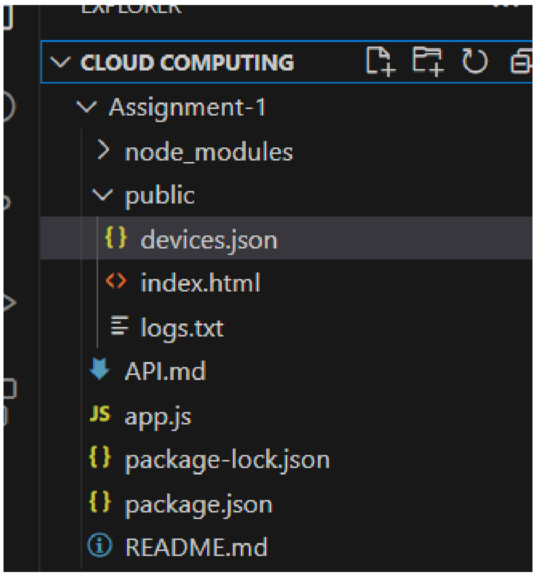


```
]
PS C:\Users\HARSHA\OneDrive\Desktop\Cloud Computing\Assignment-1> node app
Server is listening at http://localhost:3000
Existing devices are loaded from devices.json:
[
  { deviceId: '123', deviceType: 'ESP32' },
  { deviceId: '431', deviceType: 'Sensor' }
]
```



Device registered successfully

Folder structure



```
CLOUD COMPUTING
  Assignment-1
    node_modules
    public
      devices.json
      index.html
      logs.txt
    API.md
    app.js
    package-lock.json
    package.json
    README.md
```