**Department of Electrical and Computer Engineering**

**Cloud Computing EE 655/755**

Assignment #1

Node.js and Express IoT Application

Due Date: Monday, February 13, 2024

Develop a Node.js application using Express.js to manage IoT devices. The application will support registering new devices, displaying registered devices, receiving data from devices, and sending commands to devices. The system should maintain logs of all activities and persist device information.

**Device IDs and Type**

* ***Device IDs*** are typically unique identifiers assigned to each device. They can be alphanumeric strings, serial numbers, or even MAC addresses, depending on the level of uniqueness required, such as "device123", "sensor\_01", "actuatorA1".
* ***Device types*** describe the category or functionality of the device. Common examples include:
* Sensor: Devices that measure and report data, like temperature sensors, motion detectors, humidity sensors, etc. Examples: "temperature\_sensor", "motion\_detector", "humidity\_sensor".
* Actuator: Devices that perform actions or control mechanisms, like switches, valves, motors, etc. Examples: "light\_switch", "valve\_control", "motor"
* Camera: Devices used for surveillance or image capturing. Example: "security\_camera", "webcam"

In real-world applications, these values are often determined based on the hardware specifications, network architecture, and the requirements of the IoT system. They should be chosen to ensure easy identification and management of devices within the network.

**Tasks**

1. Setup and Initialization:
   1. Initialize a new Node.js project and install Express.js.
   2. Create a server listening on port 3000.
   3. Serve static files from a directory named public.
2. Device Registration:
   1. Implement a POST endpoint **/register** to register new devices.
   2. Each device should have a unique *deviceId* and a *deviceType*.
   3. Validate the presence of *deviceId* and *deviceType* in the request body.
   4. Save registered devices to a file named ***devices.json***.
   5. Respond with a status code 201 for confirming successful registration.
3. Displaying Devices:
   1. Implement a GET endpoint **/show** to display all registered devices.
   2. Read the devices from devices.json and return them in the response.
   3. Handle errors appropriately.
4. Receiving Device Data:
   1. Implement a POST endpoint **/data** for devices to send data.
   2. Ensure deviceId and data are present in the request body.
   3. Log the received data with a timestamp.
5. Sending Commands to Devices:
   1. Implement a POST endpoint **/command** to send commands to devices.
   2. Validate the presence of deviceId and command in the request body.
   3. Respond with a confirmation message and log the command with a timestamp.
6. Logging: Implement a function *logData* to log all activities (registrations, received data, commands) with timestamps to a file named logs.txt.
7. Bonus (Optional): Implement a function *loadDevices* to load existing devices from devices.json into memory on application startup.
8. Develop a Web Interface: Create a web page as a user interface for the IoT Device Management System. This interface will allow users to register new devices, send data from devices, and issue commands to devices:

A screenshot of a computer application

Description automatically generated

**Deliverables**

1. Source code of the complete application.
2. A README file with instructions on how to set up and run the application.
3. API documentation.

Submit a zip file containing all your project files. Ensure your code is well-commented and follows the best coding practices.

**Evaluation Criteria**

1. Functionality: Does the application work as intended?
2. Code Quality: Is the code well-organized, readable, and properly commented?
3. Error Handling: How effectively does the application handle errors?
4. Documentation: Clarity and completeness of the documentation.