

Week 1

A / Class Discussion Evidence

1. What is Scalability?

Definition:

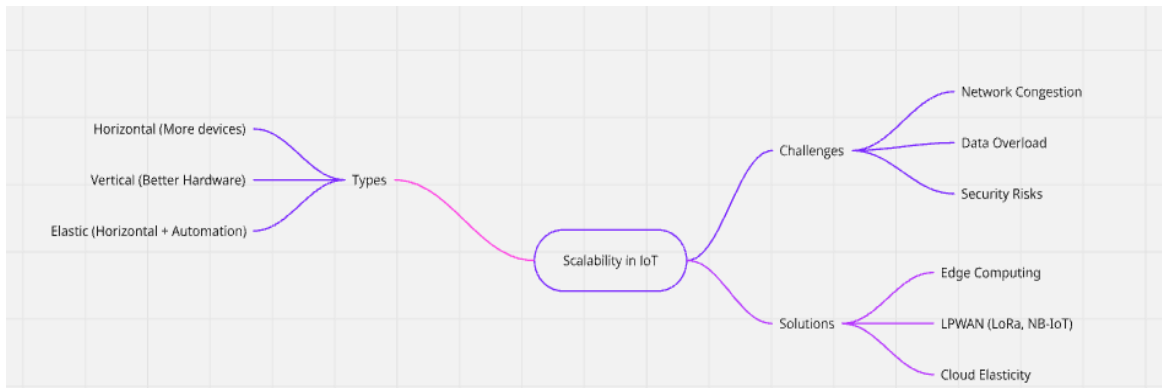
- Scalability refers to a system's ability to handle increased workload or expand its capacity without compromising performance, reliability, or efficiency.
- Key Characteristics:
 - **Horizontal Scaling:** Adding more nodes/devices (e.g., cloud servers).
 - **Vertical Scaling:** Upgrading existing hardware (e.g., CPU/RAM).
 - **Elasticity:** Dynamic scaling based on demand (common in cloud computing).

2. Scalability in Computing and IoT

- **In Computing:**
 - **Cloud Computing:** AWS/Azure auto-scaling for variable workloads.
 - **Databases:** Sharding (MongoDB) or replication (PostgreSQL).
- **In IoT:**
 - **Device Scalability:** Supporting thousands of sensors (e.g., smart cities).
 - **Network Scalability:** Protocols like MQTT, CoAP for low-overhead communication.
 - **Data Scalability:** Edge computing to reduce cloud dependency.

Why It Matters for IoT:

1. **Massive Device Growth:** 50B+ IoT devices by 2030
2. **Real-Time Processing:** Latency-critical apps (e.g., autonomous vehicles).
3. **Resource Constraints:** Limited power/budget in edge devices.



Picture 1 – A mind map show the relation of information related to Scalability in IoT.

B / Group Activity Evidence

1. What Internet-of-Things devices, applications or services do you use at home?

- Devices: Smart watch, smart fridge.
- Applications: Apple's Health app.
- Services: Tracking user info (heartbeat to measuring circadian clock, motion to detect steps, ...), detect environment light intensity (to decide whether turning up the light), detect the temperature / moisture to adjust as according.

2. What Internet-of-Things devices, applications or services do you use at university, at work, when traveling, when outside your home?

- Devices: RFID/NFC card (for entering a building / using a lift / transportation), smartphone (GPS)
- Applications: Access Management System (responsible for opening the entrance door / lift usage), Google Maps (access and collect user real-time location to send to Google Maps platform)
- Services: Backend Server / Service (for handling open door / using lift request), Google Maps platform (handling user request for route, nearby places, ...),

3. Quantify the amount of storage/data transmission/processing required for the following IoT applications.

For each, list the type of information that it collects. Examples include, key presses, favorite choices, location, distance travelled. Calculate the storage required for that application for a day/month/year.

A. Health Monitoring using a smart watch, heart rate monitor, and internet connected weight scales.

Type of Information:

Smart Watch:

- Heartbeat rate.
- Body temperature.
- Sleep / Circadian cycle.
- Accelerator (steps tracking).
- Bio-electrical impulses (ECG).
- Calories burned.

Heart Rate Monitor:

- Heartbeat rate.

Weight Scales:

- Weight.
- Body composition (fat %, muscle %, water %).

Calculated data storage information is based on DeepSeek references:

Smart Watch:

Data Type	Size / Record	Records / Day	Daily	Monthly	Yearly
Heart Rate	~ 0.1 KB	1,440	144 KB (~ 0.14 MB)	4.32 MB	~ 53.00 MB
Body temperature	~ 0.1 KB	1,440	144 KB (~ 0.14 MB)	4.32 MB	~ 53.00 MB
Steps	~ 0.05 KB	24	1.2 KB	36KB	438 KB
Sleep Data	~ 1 KB	1	1.0 KB	30 KB	365 KB
ECG	~ 5 KB	1	5 KB	150 KB	1.8 MB
Total			~ 0.3 MB	~ 9 MB	~ 108 MB

Heart Rate Monitor:

Data Type	Size / Record	Records / Day	Daily	Monthly	Yearly
Heart Rate	~ 0.1 KB	1,440	144 KB (~ 0.14 MB)	4.32 MB	~ 53.00 MB

Weight Scales

Data Type	Size / Record	Records / Day	Daily	Monthly	Yearly
Weight + Body Composition	~ 0.5 KB	1	0.5 KB	15 KB	183 KB (~ 0.18 MB)

B. Fleet management using IoT devices to track and manage vehicles in logistics and transportation.

Type of information:

- Vehicles registration number / type.
- Departure and Arrival time of a vehicle.
- Route history.
- Real-time GPS Location
- Mileage (Odometer)
- Driver identification name / number / driver license number.
- ...

Rough estimation for data storage

Data Type	Size / Record	Records / Day	Daily	Monthly	Yearly
GPS Location	~ 0.1 KB	1,440	144 KB (~ 0.14 MB)	4.32 MB	~ 53.00 MB
Travel History	~ 0.1 KB	8,640	864 KB (~ 0.85 MB)	~ 25.5 MB	~ 301.00 MB
Engine Diagnostics	~ 0.2 KB	1,44	28.8 KB	~ 864 KB (~ 0.85 MB)	~ 10.2 MB
Cargo Information	~ 0.1 KB	24	2.4 KB	72 KB	864 KB (~ 0.87 MB)
Total for 1 vehicle			~ 1.04 MB	~ 31.2 MB	~ 374.4 MB

C. Smart Agriculture using IoT devices for precision farming, crop monitoring, and livestock management.

Type of information:

- Crop:
 - Pest / disease detection.
 - Growth stage.
 - Type.
 - Weather information: air temperature, humidity, rainfall, wind speed / direction.
 - Soil: composition – pH, nutrient level, moisture level, underground temperature.
- Livestock:
 - GPS tracker.
 - Health sensor: body temperature, heartrate, feeding schedule, ID number, vaccination history.

Rough estimation for data storage:

Crop Per Hectare

Data Type	Size / Record	Records / Day	Daily Storage	Monthly	Yearly
-----------	---------------	---------------	---------------	---------	--------

Soil Information	~ 0.2 KB	24	~ 4.8 KB	~ 144 KB	~ 1,728 KB (~ 1.8 MB)
Weather Data	~ 0.2 KB	24	~ 4.8 KB	~ 144 KB	~ 1.8 MB
Total data storage per hectare			~ 10.0 KB	~ 288 KB	~ 4.0 MB

Livestock (a animal / day)

Data Type	Size / Record	Records / Day	Daily Storage	Monthly	Yearly
GPS Location	~ 0.1 KB	144	~ 14.4 KB	~ 432 KB	~ 5.26 MB
Bodily Information	~ 0.05 KB	24	~ 1.2 KB	~ 36 KB	~ 438 KB
Feeding schedule	~ 0.1 KB	24	~ 2.4 KB	~ 72 KB	~ 1.5 MB
Total data storage per animal			~ 18.4 KB	~ 600 KB (~ 0.6 MB)	~ 8.0 MB

4. Imagine a futuristic home in which everything is fully automated with Cloud-based IoT automations.

What would be the consequences if the internet service was cut off, and no local automation services exist.

A. The home air conditioning system

- Cutoff during the usage:
 - The air conditioning system will not be able to adjust the level of air flow / volume according to the changes in environment factors (moisture, air quality level, user activities ...) while “temperature” is still allowing this system to function normally as traditional thermostat does not require internet connection or it will continue to release the cool / warm air given

that the traditional thermostat system is being replaced with online automation services.

- Cutoff before any usage:
 - Like above, the system is still able to function normally thanks to thermostat given that this is a traditional thermostat instead of relying on internet connection to release or halt the release of cool / warm air. However, no local automation services exist then the air condition system is still to be able to function normally and require human intervention to manually adjust the temperature.

B. Security Access and Alarm System

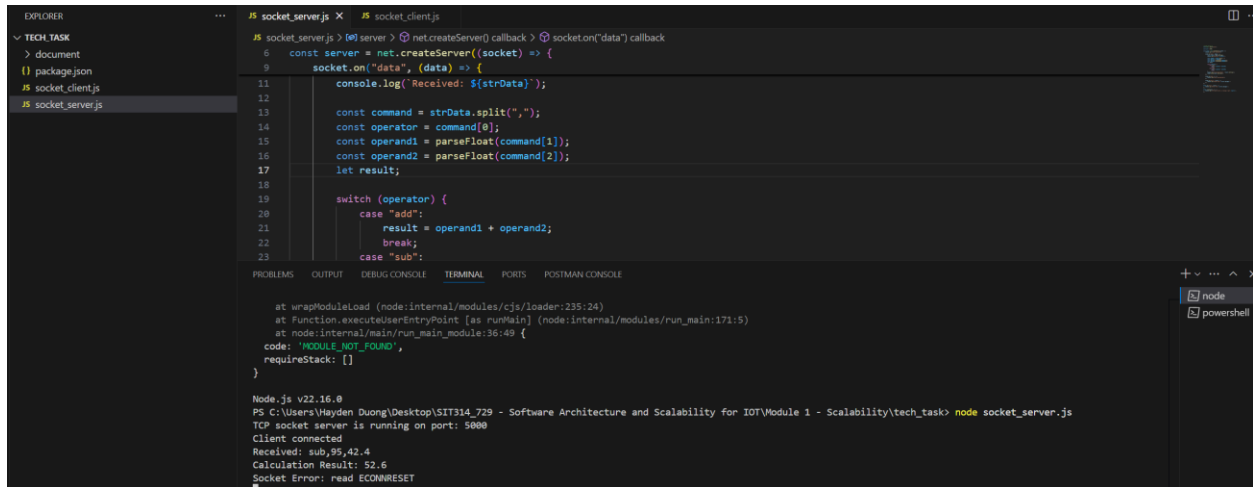
- Cutoff during the usage
 - Tenants are still able to exist the building but not be able to re-access it because Security Access is not able to send user information (access key card) to the internet to request permission to enter the premise.
 - The Alarm System is not online / working as environmental data extracted from sensors are being kept locally instead of sending to backend server for processing and issue back the appropriate commands accordingly.
- Cutoff before any usage
 - Tenants will not be able to enter the building.
 - The Alarm System in this case will share similar consequences as above.

C. Smart fire, gas, carbon monoxide monitoring system

- Cutoff during the usage:
 - The system is still able to pick up data from connecting sensors but will not be able to any adjustment accordingly as no connection is found for this system to send this information to the backend server and receive the instructions back from this latter.
- Cutoff before any usage:
 - Like above.

C / Technical Task Evidence

Testing the client-server relationship



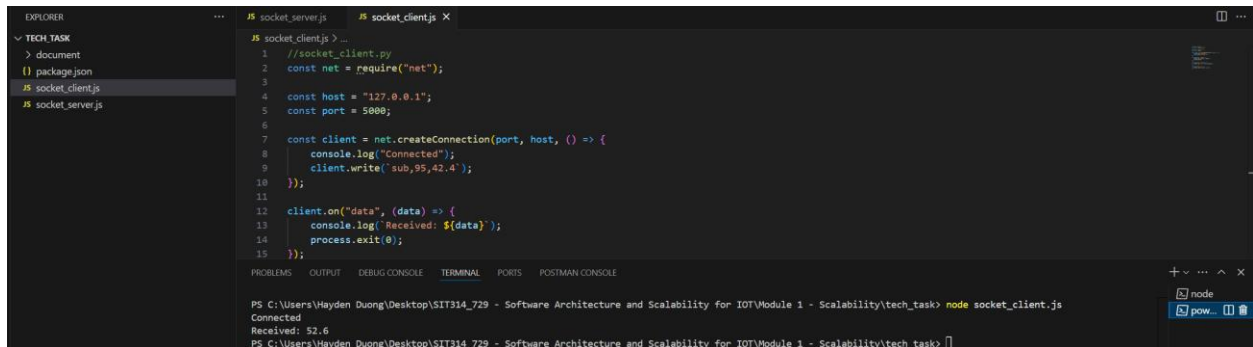
The screenshot shows the Visual Studio Code editor with the Explorer sidebar on the left. The Explorer sidebar shows a project named 'TECH_TASK' with files 'package.json', 'socket_client.js', and 'socket_server.js'. The 'socket_server.js' file is selected and its code is displayed in the main editor. The code defines a server using 'net.createServer()' and listens for data on '5000'. It parses the received data as a command and performs a calculation based on the operator. The terminal at the bottom shows the command 'node socket_server.js' being executed, and the output indicates that the server is running on port 5000 and has received a connection from 'sub,95,42.4'. The calculation result is '52.6'.

```
socket_server.js
6 const server = net.createServer((socket) => {
9   socket.on("data", (data) => {
11     console.log("Received: " + data);
12
13     const command = data.split(",");
14     const operator = command[0];
15     const operand1 = parseFloat(command[1]);
16     const operand2 = parseFloat(command[2]);
17     let result;
18
19     switch (operator) {
20       case "add":
21         result = operand1 + operand2;
22         break;
23       case "sub":
```

```
at wrapModuleLoad (node:internal/modules/cjs/loader:235:24)
at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:171:5)
at node:internal/main/run_main_module:36:49 {
  code: 'MODULE_NOT_FOUND',
  requireStack: []
}

Node.js v22.16.0
PS C:\Users\Hayden Duong\Desktop\SIIT314_729 - Software Architecture and Scalability for IOT\Module 1 - Scalability\tech_task> node socket_server.js
TCP socket server is running on port: 5000
Client connected
Received: sub,95,42.4
Calculation Result: 52.6
Socket Error: read ECONNRESET
```

Picture 1 - Server-side result.



The screenshot shows the Visual Studio Code editor with the Explorer sidebar on the left. The Explorer sidebar shows a project named 'TECH_TASK' with files 'package.json', 'socket_client.js', and 'socket_server.js'. The 'socket_client.js' file is selected and its code is displayed in the main editor. The code defines a client using 'net.createConnection()' and connects to '127.0.0.1' on port '5000'. It sends a command 'sub,95,42.4' to the server. The terminal at the bottom shows the command 'node socket_client.js' being executed, and the output indicates that the client is connected and has received the calculation result '52.6'.

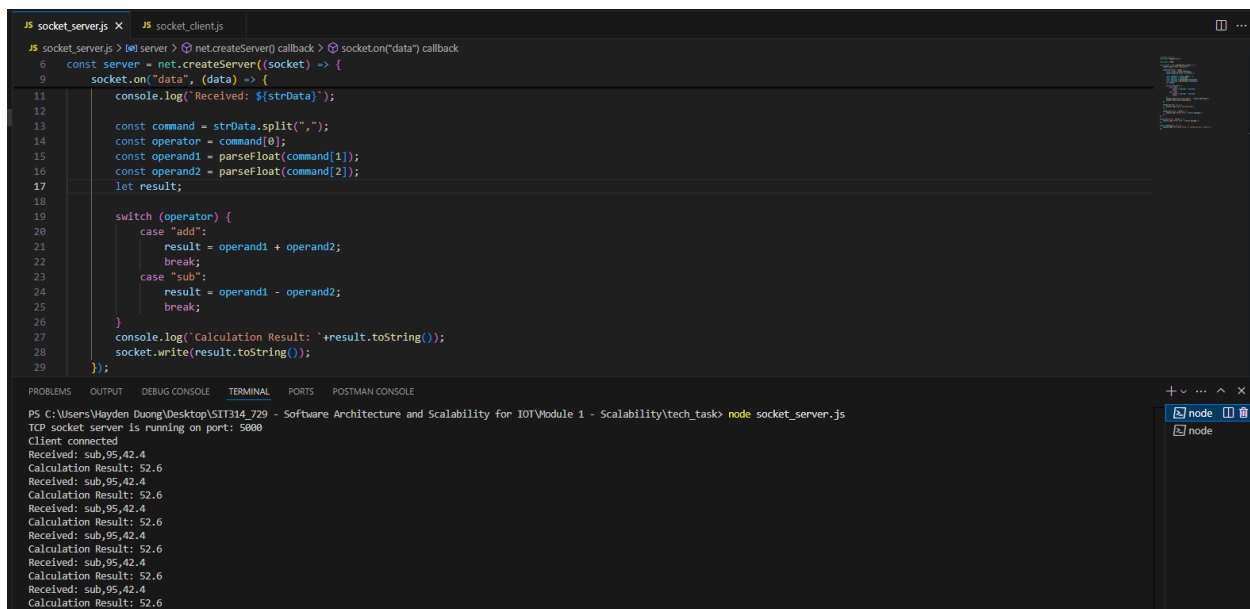
```
socket_client.js
1 //socket_client.py
2 const net = require("net");
3
4 const host = "127.0.0.1";
5 const port = 5000;
6
7 const client = net.createConnection(port, host, () => {
8   console.log("Connected");
9   client.write("sub,95,42.4");
10 });
11
12 client.on("data", (data) => {
13   console.log("Received: " + data);
14   process.exit(0);
15 });
```

```
PS C:\Users\Hayden Duong\Desktop\SIIT314_729 - Software Architecture and Scalability for IOT\Module 1 - Scalability\tech_task> node socket_client.js
Connected
Received: 52.6
PS C:\Users\Hayden Duong\Desktop\SIIT314_729 - Software Architecture and Scalability for IOT\Module 1 - Scalability\tech_task>
```

Picture 2 – Client-side result.

Experiment with code

A/ The interval of sending the data (repeat the comment every 2 seconds)



```
JS socket_server.js X JS socket_client.js
JS socket_server.js > net.createServer() callback > socket.on("data") callback
6 const server = net.createServer((socket) => {
9   socket.on("data", (data) => {
11     console.log("Received: ${strData}");
12
13     const command = strData.split(",");
14     const operator = command[0];
15     const operand1 = parseFloat(command[1]);
16     const operand2 = parseFloat(command[2]);
17     let result;
18
19     switch (operator) {
20       case "add":
21         result = operand1 + operand2;
22         break;
23       case "sub":
24         result = operand1 - operand2;
25         break;
26     }
27     console.log("Calculation Result: "+result.toString());
28     socket.write(result.toString());
29   });
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

PS C:\Users\Hayden Duong\Desktop\SIT314_729 - Software Architecture and Scalability for IOT\Module 1 - Scalability\tech_task> node socket_server.js

TCP socket server is running on port: 5000

Client connected

Received: sub,95,42.4

Calculation Result: 52.6

Received: sub,95,42.4

Calculation Result: 52.6

Received: sub,95,42.4

Calculation Result: 52.6

Received: sub,95,42.4

Calculation Result: 52.6

Received: sub,95,42.4

Calculation Result: 52.6

Received: sub,95,42.4

Calculation Result: 52.6

Received: sub,95,42.4

Calculation Result: 52.6

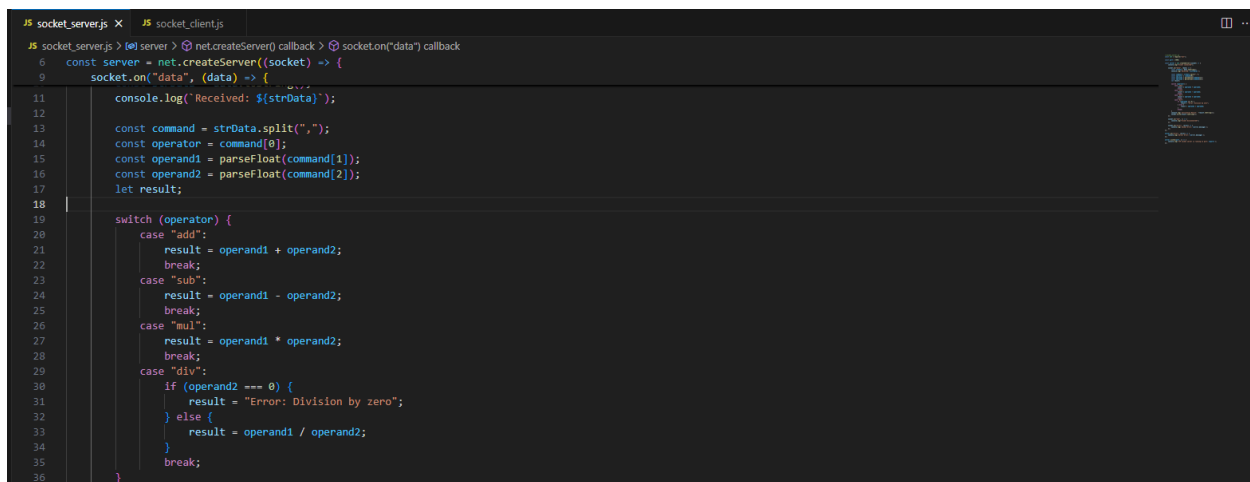
Picture 3 – Result from Server-side.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
PS C:\Users\Hayden Duong\Desktop\SIT314_729 - Software Architecture and Scalability for IOT\Module 1 - Scalability\tech_task> node socket_client.js
Connected
Received: 52.6
Received: 52.6
Received: 52.6
Received: 52.6
Received: 52.6
Received: 52.6
Received: 52.6
Received: 52.6
Received: 52.6
Received: 52.6
Received: 52.6
Received: 52.6
Error: read ECONNRESET
Connection closed
[]
```

Picture 4 – Result from Client-side.

B/ Modify the server to support different calculations



```
JS socket_server.js X JS socket_client.js
JS socket_server.js > net.createServer() callback > socket.on("data") callback
6 const server = net.createServer((socket) => {
9   socket.on("data", (data) => {
11     console.log("Received: ${strData}");
12
13     const command = strData.split(",");
14     const operator = command[0];
15     const operand1 = parseFloat(command[1]);
16     const operand2 = parseFloat(command[2]);
17     let result;
18
19     switch (operator) {
20       case "add":
21         result = operand1 + operand2;
22         break;
23       case "sub":
24         result = operand1 - operand2;
25         break;
26       case "mul":
27         result = operand1 * operand2;
28         break;
29       case "div":
30         if (operand2 === 0) {
31           result = "Error: Division by zero";
32         } else {
33           result = operand1 / operand2;
34         }
35         break;
36     }
37   });
}
```

Picture 5 – Code-addition to socket_server.js

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
PS C:\Users\Hayden Duong\Desktop\SIIT314_729 - Software Architecture and Scalability for IoT\Module 1 - Scalability\tech_task> node socket_server.js
TCP socket server is running on port: 5000
Client connected
Received: mul,2,3
Calculation Result: 6
Socket Error: read ECONNRESET
[]
```

Picture 6 – Result from Server-side

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
PS C:\Users\Hayden Duong\Desktop\SIIT314_729 - Software Architecture and Scalability for IoT\Module 1 - Scalability\tech_task> node socket_client.js
Connected
Received: 6
PS C:\Users\Hayden Duong\Desktop\SIIT314_729 - Software Architecture and Scalability for IoT\Module 1 - Scalability\tech_task> []
```

Picture 7 – Result from Client-side

C/ Modify the client code so it reads a random number and sends it as part of the calculation

```
JS socket_server.js JS socket_client.js X
JS socket_client.js > client > net.createConnection() callback
1 //socket_client.py
2 const { randomInt } = require("crypto");
3 const net = require("net");
4
5 const host = "127.0.0.1";
6 const port = 5000;
7
8 const client = net.createConnection(port, host, () => {
9   console.log("Connected");
10
11   const a = randomInt(0, 10);
12   const b = randomInt(0, 10);
13   const commands = ["add", "sub", "mul", "div"];
14   const operator = commands[randomInt(0, commands.length)];
15
16   console.log(`Random numbers generated: ${a}, ${b}`);
17   console.log(`Operator selected: ${operator}`);
18
19   // client.write("add,$(a),$(b)");
20   // client.write("sub,$(a),$(b)");
21   // client.write("mul,$(a),$(b)");
22   // client.write("div,$(a),$(b)");
23
24   client.write(`${operator},${a},${b}`);
25 });
```

Picture 8 – Code addition & modification to socket_client.js

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
PS C:\Users\Hayden Duong\Desktop\SIIT314_729 - Software Architecture and Scalability for IoT\Module 1 - Scalability\tech_task> node socket_server.js
TCP socket server is running on port: 5000
Client connected
Received: add,6,5
Calculation Result: 11
Socket Error: read ECONNRESET
[]
```

Picture 9 – Result from Server-side

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
PS C:\Users\Hayden Duong\Desktop\SIT314_729 - Software Architecture and Scalability for IOT\Module 1 - Scalability\tech_task> node socket_client.js
Connected
Random numbers generated: 6, 5
Operator selected: add
Received: 11
```

Picture 10 – Result from Client-side

D/ Number of clients

```
JS socket_server.js JS socket_client.js JS stress_test.js X
JS stress_test.js > ...
1 const { randomInt } = require("crypto");
2 const net = require("net");
3
4 const host = "127.0.0.1";
5 const port = 5000;
6 const numClients = 10; // 50 client giả lập
7
8 for (let i = 0; i < numClients; i++) {
9   const client = net.createConnection(port, host, () => {
10     const a = randomInt(0, 10);
11     const b = randomInt(0, 10);
12     const operator = ["add", "sub", "mul", "div"][randomInt(0, 4)];
13
14     client.write(`${operator},${a},${b}\n`);
15     console.log(`Client ${i + 1} sent: ${operator},${a},${b}`);
16   });
17
18   client.on("data", (data) => {
19     console.log(`Client ${i + 1} received: ${data.toString().trim()}`);
20     client.end();
21   });
22 }
```

Picture 11 – Created a new JS file called stress_test.js to simulate several clients are connecting to socket_server.js (code suggested by DeepSeek)

```
PS C:\Users\Hayden Duong\Desktop\SIT314_729 - Software Architecture and Scalability for IOT\Module 1 - Scalability\tech_task> node socket_server.js
TCP socket server is running on port: 5000
Client 1 sent: add,8,2
Calculation Result: 10

Client 2 sent: mul,9,9
Calculation Result: 81

Client 3 sent: add,0,8
Calculation Result: 8

Client 4 sent: div,4,7
Calculation Result: 0.5714285714285714

Client 5 sent: mul,8,3
Calculation Result: 24

Client 6 sent: mul,3,0
Calculation Result: 0

Client 7 sent: sub,9,8
Calculation Result: 1

Client 8 sent: div,2,2
Calculation Result: 1

Client 9 sent: mul,2,1
Calculation Result: 2

Client 10 sent: sub,4,5
Calculation Result: -1

Client 1 disconnected
Client 2 disconnected
Client 3 disconnected
Client 4 disconnected
Client 5 disconnected
Client 6 disconnected
Client 7 disconnected
Client 8 disconnected
Client 9 disconnected
Client 10 disconnected
```

Picture 12 – Result from Server-side

```
PS C:\Users\Hayden Duong\Desktop\SIT314_729 - Software Architecture and Scalability for IOT\Module 1 - Scalability\tech_task> node stress_test.js
Client 1 sent: add,8,2 | received: 10

Client 2 sent: mul,9,9 | received: 81

Client 3 sent: add,0,8 | received: 8

Client 4 sent: div,4,7 | received: 0.5714285714285714

Client 5 sent: mul,8,3 | received: 24

Client 6 sent: mul,3,0 | received: 0

Client 7 sent: sub,9,8 | received: 1

Client 8 sent: div,2,2 | received: 1

Client 9 sent: mul,2,1 | received: 2

Client 10 sent: sub,4,5 | received: -1

PS C:\Users\Hayden Duong\Desktop\SIT314_729 - Software Architecture and Scalability for IOT\Module 1 - Scalability\tech_task> 
```

Picture 13 – Result from Client-side

E/ Connect between other computers (need to share stress_test.js for other group members to achieve different IP addresses display in the server-side terminal – tested with)

```
JS socket_server.js X JS stress_test.js
JS socket_server.js > [0] server > [0] net.createServer() callback > [0] socket.on("data") callback
1 //socket_server.py
2 const net = require("net");
3
4 const port = 5000;
5
6 let clientCount = 0;
7
8 const server = net.createServer((socket) => {
9
10     const clientIP = socket.remoteAddress;
11     const clientId = ++clientCount;
12
13     socket.on("data", (data) => {
14         const strData = data.toString().trim();
15         console.log(`Client ${clientId} connected from: ${clientIP} | sent: ${strData}`);
16
17         const command = strData.split(",");
18         const operator = command[0];
19     });
20 });
21
22 server.listen(port, () => {
23     console.log(`Server is running on port: ${port}`);
24 });
25
26 server.on("close", () => {
27     console.log("Server closed");
28 });
29
30 server.on("error", (err) => {
31     console.log("Server error: ", err);
32 });
33
34 server.on("clientError", (err, socket) => {
35     console.log("Client error: ", err);
36     socket.destroy();
37 });
38
39 server.on("connection", (socket) => {
40     console.log("New client connected");
41 });
42
43 server.on("disconnect", (socket) => {
44     console.log("Client disconnected");
45 });
46
47 server.on("close", () => {
48     console.log("Server closed");
49 });
50
51 server.on("error", (err) => {
52     console.log("Server error: ", err);
53 });
54
55 server.on("clientError", (err, socket) => {
56     console.log("Client error: ", err);
57     socket.destroy();
58 });
59
60 server.on("connection", (socket) => {
61     console.log("New client connected");
62 });
63
64 server.on("disconnect", (socket) => {
65     console.log("Client disconnected");
66 });
67
68 server.on("close", () => {
69     console.log("Server closed");
70 });
71
72 server.on("error", (err) => {
73     console.log("Server error: ", err);
74 });
75
76 server.on("clientError", (err, socket) => {
77     console.log("Client error: ", err);
78     socket.destroy();
79 });
80
81 server.on("connection", (socket) => {
82     console.log("New client connected");
83 });
84
85 server.on("disconnect", (socket) => {
86     console.log("Client disconnected");
87 });
88
89 server.on("close", () => {
90     console.log("Server closed");
91 });
92
93 server.on("error", (err) => {
94     console.log("Server error: ", err);
95 });
96
97 server.on("clientError", (err, socket) => {
98     console.log("Client error: ", err);
99     socket.destroy();
100 });
101
102 server.on("connection", (socket) => {
103     console.log("New client connected");
104 });
105
106 server.on("disconnect", (socket) => {
107     console.log("Client disconnected");
108 });
109
110 server.on("close", () => {
111     console.log("Server closed");
112 });
113
114 server.on("error", (err) => {
115     console.log("Server error: ", err);
116 });
117
118 server.on("clientError", (err, socket) => {
119     console.log("Client error: ", err);
120     socket.destroy();
121 });
122
123 server.on("connection", (socket) => {
124     console.log("New client connected");
125 });
126
127 server.on("disconnect", (socket) => {
128     console.log("Client disconnected");
129 });
130
131 server.on("close", () => {
132     console.log("Server closed");
133 });
134
135 server.on("error", (err) => {
136     console.log("Server error: ", err);
137 });
138
139 server.on("clientError", (err, socket) => {
140     console.log("Client error: ", err);
141     socket.destroy();
142 });
143
144 server.on("connection", (socket) => {
145     console.log("New client connected");
146 });
147
148 server.on("disconnect", (socket) => {
149     console.log("Client disconnected");
150 });
151
152 server.on("close", () => {
153     console.log("Server closed");
154 });
155
156 server.on("error", (err) => {
157     console.log("Server error: ", err);
158 });
159
160 server.on("clientError", (err, socket) => {
161     console.log("Client error: ", err);
162     socket.destroy();
163 });
164
165 server.on("connection", (socket) => {
166     console.log("New client connected");
167 });
168
169 server.on("disconnect", (socket) => {
170     console.log("Client disconnected");
171 });
172
173 server.on("close", () => {
174     console.log("Server closed");
175 });
176
177 server.on("error", (err) => {
178     console.log("Server error: ", err);
179 });
180
181 server.on("clientError", (err, socket) => {
182     console.log("Client error: ", err);
183     socket.destroy();
184 });
185
186 server.on("connection", (socket) => {
187     console.log("New client connected");
188 });
189
190 server.on("disconnect", (socket) => {
191     console.log("Client disconnected");
192 });
193
194 server.on("close", () => {
195     console.log("Server closed");
196 });
197
198 server.on("error", (err) => {
199     console.log("Server error: ", err);
200 });
201
202 server.on("clientError", (err, socket) => {
203     console.log("Client error: ", err);
204     socket.destroy();
205 });
206
207 server.on("connection", (socket) => {
208     console.log("New client connected");
209 });
210
211 server.on("disconnect", (socket) => {
212     console.log("Client disconnected");
213 });
214
215 server.on("close", () => {
216     console.log("Server closed");
217 });
218
219 server.on("error", (err) => {
220     console.log("Server error: ", err);
221 });
222
223 server.on("clientError", (err, socket) => {
224     console.log("Client error: ", err);
225     socket.destroy();
226 });
227
228 server.on("connection", (socket) => {
229     console.log("New client connected");
230 });
231
232 server.on("disconnect", (socket) => {
233     console.log("Client disconnected");
234 });
235
236 server.on("close", () => {
237     console.log("Server closed");
238 });
239
240 server.on("error", (err) => {
241     console.log("Server error: ", err);
242 });
243
244 server.on("clientError", (err, socket) => {
245     console.log("Client error: ", err);
246     socket.destroy();
247 });
248
249 server.on("connection", (socket) => {
250     console.log("New client connected");
251 });
252
253 server.on("disconnect", (socket) => {
254     console.log("Client disconnected");
255 });
256
257 server.on("close", () => {
258     console.log("Server closed");
259 });
260
261 server.on("error", (err) => {
262     console.log("Server error: ", err);
263 });
264
265 server.on("clientError", (err, socket) => {
266     console.log("Client error: ", err);
267     socket.destroy();
268 });
269
270 server.on("connection", (socket) => {
271     console.log("New client connected");
272 });
273
274 server.on("disconnect", (socket) => {
275     console.log("Client disconnected");
276 });
277
278 server.on("close", () => {
279     console.log("Server closed");
280 });
281
282 server.on("error", (err) => {
283     console.log("Server error: ", err);
284 });
285
286 server.on("clientError", (err, socket) => {
287     console.log("Client error: ", err);
288     socket.destroy();
289 });
290
291 server.on("connection", (socket) => {
292     console.log("New client connected");
293 });
294
295 server.on("disconnect", (socket) => {
296     console.log("Client disconnected");
297 });
298
299 server.on("close", () => {
300     console.log("Server closed");
301 });
302
303 server.on("error", (err) => {
304     console.log("Server error: ", err);
305 });
306
307 server.on("clientError", (err, socket) => {
308     console.log("Client error: ", err);
309     socket.destroy();
310 });
311
312 server.on("connection", (socket) => {
313     console.log("New client connected");
314 });
315
316 server.on("disconnect", (socket) => {
317     console.log("Client disconnected");
318 });
319
320 server.on("close", () => {
321     console.log("Server closed");
322 });
323
324 server.on("error", (err) => {
325     console.log("Server error: ", err);
326 });
327
328 server.on("clientError", (err, socket) => {
329     console.log("Client error: ", err);
330     socket.destroy();
331 });
332
333 server.on("connection", (socket) => {
334     console.log("New client connected");
335 });
336
337 server.on("disconnect", (socket) => {
338     console.log("Client disconnected");
339 });
340
341 server.on("close", () => {
342     console.log("Server closed");
343 });
344
345 server.on("error", (err) => {
346     console.log("Server error: ", err);
347 });
348
349 server.on("clientError", (err, socket) => {
350     console.log("Client error: ", err);
351     socket.destroy();
352 });
353
354 server.on("connection", (socket) => {
355     console.log("New client connected");
356 });
357
358 server.on("disconnect", (socket) => {
359     console.log("Client disconnected");
360 });
361
362 server.on("close", () => {
363     console.log("Server closed");
364 });
365
366 server.on("error", (err) => {
367     console.log("Server error: ", err);
368 });
369
370 server.on("clientError", (err, socket) => {
371     console.log("Client error: ", err);
372     socket.destroy();
373 });
374
375 server.on("connection", (socket) => {
376     console.log("New client connected");
377 });
378
379 server.on("disconnect", (socket) => {
380     console.log("Client disconnected");
381 });
382
383 server.on("close", () => {
384     console.log("Server closed");
385 });
386
387 server.on("error", (err) => {
388     console.log("Server error: ", err);
389 });
390
391 server.on("clientError", (err, socket) => {
392     console.log("Client error: ", err);
393     socket.destroy();
394 });
395
396 server.on("connection", (socket) => {
397     console.log("New client connected");
398 });
399
400 server.on("disconnect", (socket) => {
401     console.log("Client disconnected");
402 });
403
404 server.on("close", () => {
405     console.log("Server closed");
406 });
407
408 server.on("error", (err) => {
409     console.log("Server error: ", err);
410 });
411
412 server.on("clientError", (err, socket) => {
413     console.log("Client error: ", err);
414     socket.destroy();
415 });
416
417 server.on("connection", (socket) => {
418     console.log("New client connected");
419 });
420
421 server.on("disconnect", (socket) => {
422     console.log("Client disconnected");
423 });
424
425 server.on("close", () => {
426     console.log("Server closed");
427 });
428
429 server.on("error", (err) => {
430     console.log("Server error: ", err);
431 });
432
433 server.on("clientError", (err, socket) => {
434     console.log("Client error: ", err);
435     socket.destroy();
436 });
437
438 server.on("connection", (socket) => {
439     console.log("New client connected");
440 });
441
442 server.on("disconnect", (socket) => {
443     console.log("Client disconnected");
444 });
445
446 server.on("close", () => {
447     console.log("Server closed");
448 });
449
450 server.on("error", (err) => {
451     console.log("Server error: ", err);
452 });
453
454 server.on("clientError", (err, socket) => {
455     console.log("Client error: ", err);
456     socket.destroy();
457 });
458
459 server.on("connection", (socket) => {
460     console.log("New client connected");
461 });
462
463 server.on("disconnect", (socket) => {
464     console.log("Client disconnected");
465 });
466
467 server.on("close", () => {
468     console.log("Server closed");
469 });
470
471 server.on("error", (err) => {
472     console.log("Server error: ", err);
473 });
474
475 server.on("clientError", (err, socket) => {
476     console.log("Client error: ", err);
477     socket.destroy();
478 });
479
480 server.on("connection", (socket) => {
481     console.log("New client connected");
482 });
483
484 server.on("disconnect", (socket) => {
485     console.log("Client disconnected");
486 });
487
488 server.on("close", () => {
489     console.log("Server closed");
490 });
491
492 server.on("error", (err) => {
493     console.log("Server error: ", err);
494 });
495
496 server.on("clientError", (err, socket) => {
497     console.log("Client error: ", err);
498     socket.destroy();
499 });
500
501 server.on("connection", (socket) => {
502     console.log("New client connected");
503 });
504
505 server.on("disconnect", (socket) => {
506     console.log("Client disconnected");
507 });
508
509 server.on("close", () => {
510     console.log("Server closed");
511 });
512
513 server.on("error", (err) => {
514     console.log("Server error: ", err);
515 });
516
517 server.on("clientError", (err, socket) => {
518     console.log("Client error: ", err);
519     socket.destroy();
520 });
521
522 server.on("connection", (socket) => {
523     console.log("New client connected");
524 });
525
526 server.on("disconnect", (socket) => {
527     console.log("Client disconnected");
528 });
529
530 server.on("close", () => {
531     console.log("Server closed");
532 });
533
534 server.on("error", (err) => {
535     console.log("Server error: ", err);
536 });
537
538 server.on("clientError", (err, socket) => {
539     console.log("Client error: ", err);
540     socket.destroy();
541 });
542
543 server.on("connection", (socket) => {
544     console.log("New client connected");
545 });
546
547 server.on("disconnect", (socket) => {
548     console.log("Client disconnected");
549 });
550
551 server.on("close", () => {
552     console.log("Server closed");
553 });
554
555 server.on("error", (err) => {
556     console.log("Server error: ", err);
557 });
558
559 server.on("clientError", (err, socket) => {
560     console.log("Client error: ", err);
561     socket.destroy();
562 });
563
564 server.on("connection", (socket) => {
565     console.log("New client connected");
566 });
567
568 server.on("disconnect", (socket) => {
569     console.log("Client disconnected");
570 });
571
572 server.on("close", () => {
573     console.log("Server closed");
574 });
575
576 server.on("error", (err) => {
577     console.log("Server error: ", err);
578 });
579
580 server.on("clientError", (err, socket) => {
581     console.log("Client error: ", err);
582     socket.destroy();
583 });
584
585 server.on("connection", (socket) => {
586     console.log("New client connected");
587 });
588
589 server.on("disconnect", (socket) => {
590     console.log("Client disconnected");
591 });
592
593 server.on("close", () => {
594     console.log("Server closed");
595 });
596
597 server.on("error", (err) => {
598     console.log("Server error: ", err);
599 });
600
601 server.on("clientError", (err, socket) => {
602     console.log("Client error: ", err);
603     socket.destroy();
604 });
605
606 server.on("connection", (socket) => {
607     console.log("New client connected");
608 });
609
610 server.on("disconnect", (socket) => {
611     console.log("Client disconnected");
612 });
613
614 server.on("close", () => {
615     console.log("Server closed");
616 });
617
618 server.on("error", (err) => {
619     console.log("Server error: ", err);
620 });
621
622 server.on("clientError", (err, socket) => {
623     console.log("Client error: ", err);
624     socket.destroy();
625 });
626
627 server.on("connection", (socket) => {
628     console.log("New client connected");
629 });
630
631 server.on("disconnect", (socket) => {
632     console.log("Client disconnected");
633 });
634
635 server.on("close", () => {
636     console.log("Server closed");
637 });
638
639 server.on("error", (err) => {
640     console.log("Server error: ", err);
641 });
642
643 server.on("clientError", (err, socket) => {
644     console.log("Client error: ", err);
645     socket.destroy();
646 });
647
648 server.on("connection", (socket) => {
649     console.log("New client connected");
650 });
651
652 server.on("disconnect", (socket) => {
653     console.log("Client disconnected");
654 });
655
656 server.on("close", () => {
657     console.log("Server closed");
658 });
659
660 server.on("error", (err) => {
661     console.log("Server error: ", err);
662 });
663
664 server.on("clientError", (err, socket) => {
665     console.log("Client error: ", err);
666     socket.destroy();
667 });
668
669 server.on("connection", (socket) => {
670     console.log("New client connected");
671 });
672
673 server.on("disconnect", (socket) => {
674     console.log("Client disconnected");
675 });
676
677 server.on("close", () => {
678     console.log("Server closed");
679 });
680
681 server.on("error", (err) => {
682     console.log("Server error: ", err);
683 });
684
685 server.on("clientError", (err, socket) => {
686     console.log("Client error: ", err);
687     socket.destroy();
688 });
689
690 server.on("connection", (socket) => {
691     console.log("New client connected");
692 });
693
694 server.on("disconnect", (socket) => {
695     console.log("Client disconnected");
696 });
697
698 server.on("close", () => {
699     console.log("Server closed");
700 });
701
702 server.on("error", (err) => {
703     console.log("Server error: ", err);
704 });
705
706 server.on("clientError", (err, socket) => {
707     console.log("Client error: ", err);
708     socket.destroy();
709 });
710
711 server.on("connection", (socket) => {
712     console.log("New client connected");
713 });
714
715 server.on("disconnect", (socket) => {
716     console.log("Client disconnected");
717 });
718
719 server.on("close", () => {
720     console.log("Server closed");
721 });
722
723 server.on("error", (err) => {
724     console.log("Server error: ", err);
725 });
726
727 server.on("clientError", (err, socket) => {
728     console.log("Client error: ", err);
729     socket.destroy();
730 });
731
732 server.on("connection", (socket) => {
733     console.log("New client connected");
734 });
735
736 server.on("disconnect", (socket) => {
737     console.log("Client disconnected");
738 });
739
740 server.on("close", () => {
741     console.log("Server closed");
742 });
743
744 server.on("error", (err) => {
745     console.log("Server error: ", err);
746 });
747
748 server.on("clientError", (err, socket) => {
749     console.log("Client error: ", err);
750     socket.destroy();
751 });
752
753 server.on("connection", (socket) => {
754     console.log("New client connected");
755 });
756
757 server.on("disconnect", (socket) => {
758     console.log("Client disconnected");
759 });
760
761 server.on("close", () => {
762     console.log("Server closed");
763 });
764
765 server.on("error", (err) => {
766     console.log("Server error: ", err);
767 });
768
769 server.on("clientError", (err, socket) => {
770     console.log("Client error: ", err);
771     socket.destroy();
772 });
773
774 server.on("connection", (socket) => {
775     console.log("New client connected");
776 });
777
778 server.on("disconnect", (socket) => {
779     console.log("Client disconnected");
780 });
781
782 server.on("close", () => {
783     console.log("Server closed");
784 });
785
786 server.on("error", (err) => {
787     console.log("Server error: ", err);
788 });
789
790 server.on("clientError", (err, socket) => {
791     console.log("Client error: ", err);
792     socket.destroy();
793 });
794
795 server.on("connection", (socket) => {
796     console.log("New client connected");
797 });
798
799 server.on("disconnect", (socket) => {
800     console.log("Client disconnected");
801 });
802
803 server.on("close", () => {
804     console.log("Server closed");
805 });
806
807 server.on("error", (err) => {
808     console.log("Server error: ", err);
809 });
810
811 server.on("clientError", (err, socket) => {
812     console.log("Client error: ", err);
813     socket.destroy();
814 });
815
816 server.on("connection", (socket) => {
817     console.log("New client connected");
818 });
819
820 server.on("disconnect", (socket) => {
821     console.log("Client disconnected");
822 });
823
824 server.on("close", () => {
825     console.log("Server closed");
826 });
827
828 server.on("error", (err) => {
829     console.log("Server error: ", err);
830 });
831
832 server.on("clientError", (err, socket) => {
833     console.log("Client error: ", err);
834     socket.destroy();
835 });
836
837 server.on("connection", (socket) => {
838     console.log("New client connected");
839 });
840
841 server.on("disconnect", (socket) => {
842     console.log("Client disconnected");
843 });
844
845 server.on("close", () => {
846     console.log("Server closed");
847 });
848
849 server.on("error", (err) => {
850     console.log("Server error: ", err);
851 });
852
853 server.on("clientError", (err, socket) => {
854     console.log("Client error: ", err);
855     socket.destroy();
856 });
857
858 server.on("connection", (socket) => {
859     console.log("New client connected");
860 });
861
862 server.on("disconnect", (socket) => {
863     console.log("Client disconnected");
864 });
865
866 server.on("close", () => {
867     console.log("Server closed");
868 });
869
870 server.on("error", (err) => {
871     console.log("Server error: ", err);
872 });
873
874 server.on("clientError", (err, socket) => {
875     console.log("Client error: ", err);
876     socket.destroy();
877 });
878
879 server.on("connection", (socket) => {
880     console.log("New client connected");
881 });
882
883 server.on("disconnect", (socket) => {
884     console.log("Client disconnected");
885 });
886
887 server.on("close", () => {
888     console.log("Server closed");
889 });
890
891 server.on("error", (err) => {
892     console.log("Server error: ", err);
893 });
894
895 server.on("clientError", (err, socket) => {
896     console.log("Client error: ", err);
897     socket.destroy();
898 });
899
900 server.on("connection", (socket) => {
901     console.log("New client connected");
902 });
903
904 server.on("disconnect", (socket) => {
905     console.log("Client disconnected");
906 });
907
908 server.on("close", () => {
909     console.log("Server closed");
910 });
911
912 server.on("error", (err) => {
913     console.log("Server error: ", err);
914 });
915
916 server.on("clientError", (err, socket) => {
917     console.log("Client error: ", err);
918     socket.destroy();
919 });
920
921 server.on("connection", (socket) => {
922     console.log("New client connected");
923 });
924
925 server.on("disconnect", (socket) => {
926     console.log("Client disconnected");
927 });
928
929 server.on("close", () => {
930     console.log("Server closed");
931 });
932
933 server.on("error", (err) => {
934     console.log("Server error: ", err);
935 });
936
937 server.on("clientError", (err, socket) => {
938     console.log("Client error: ", err);
939     socket.destroy();
940 });
941
942 server.on("connection", (socket) => {
943     console.log("New client connected");
944 });
945
946 server.on("disconnect", (socket) => {
947     console.log("Client disconnected");
948 });
949
950 server.on("close", () => {
951     console.log("Server closed");
952 });
953
954 server.on("error", (err) => {
955     console.log("Server error: ", err);
956 });
957
958 server.on("clientError", (err, socket) => {
959     console.log("Client error: ", err);
960     socket.destroy();
961 });
962
963 server.on("connection", (socket) => {
964     console.log("New client connected");
965 });
966
967 server.on("disconnect", (socket) => {
968     console.log("Client disconnected");
969 });
970
971 server.on("close", () => {
972     console.log("Server closed");
973 });
974
975 server.on("error", (err) => {
976     console.log("Server error: ", err);
977 });
978
979 server.on("clientError", (err, socket) => {
980     console.log("Client error: ", err);
981     socket.destroy();
982 });
983
984 server.on("connection", (socket) => {
985     console.log("New client connected");
986 });
987
988 server.on("disconnect", (socket) => {
989     console.log("Client disconnected");
990 });
991
992 server.on("close", () => {
993     console.log("Server closed");
994 });
995
996 server.on("error", (err) => {
997     console.log("Server error: ", err);
998 });
999
1000 server.on("clientError", (err, socket) => {
1001     console.log("Client error: ", err);
1002     socket.destroy();
1003 });
1004
1005 server.on("connection", (socket) => {
1006     console.log("New client connected");
1007 });
1008
1009 server.on("disconnect", (socket) => {
1010     console.log("Client disconnected");
1011 });
1012
1013 server.on("close", () => {
1014     console.log("Server closed");
1015 });
1016
1017 server.on("error", (err) => {
1018     console.log("Server error: ", err);
1019 });
1020
1021 server.on("clientError", (err, socket) => {
1022     console.log("Client error: ", err);
1023     socket.destroy();
1024 });
1025
1026 server.on("connection", (socket) => {
1027     console.log("New client connected");
1028 });
1029
1030 server.on("disconnect", (socket) => {
1031     console.log("Client disconnected");
1032 });
1033
1034 server.on("close", () => {
1035     console.log("Server closed");
1036 });
1037
1038 server.on("error", (err) => {
1039     console.log("Server error: ", err);
1040 });
1041
1042 server.on("clientError", (err, socket) => {
1043     console.log("Client error: ", err);
1044     socket.destroy();
1045 });
1046
1047 server.on("connection", (socket) => {
1048     console.log("New client connected");
1049 });
1050
1051 server.on("disconnect", (socket) => {
1052     console.log("Client disconnected");
1053 });
1054
1055 server.on("close", () => {
1056     console.log("Server closed");
1057 });
1058
1059 server.on("error", (err) => {
1060     console.log("Server error: ", err);
1061 });
1062
1063 server.on("clientError", (err, socket) => {
1064     console.log("Client error: ", err);
1065     socket.destroy();
1066 });
1067
1068 server.on("connection", (socket) => {
1069     console.log("New client connected");
1070 });
1071
1072 server.on("disconnect", (socket) => {
1073     console.log("Client disconnected");
1074 });
1075
1076 server.on("close", () => {
1077     console.log("Server closed");
1078 });
1079
1080 server.on("error", (err) => {
1081     console.log("Server error: ", err);
1082 });
1083
1084 server.on("clientError", (err, socket) => {
1085     console.log("Client error: ", err);
1086     socket.destroy();
1087 });
1088
1089 server.on("connection", (socket) => {
1090     console.log("New client connected");
1091 });
1092
1093 server.on("disconnect", (socket) => {
1094     console.log("Client disconnected");
1095 });
1096
1097 server.on("close", () => {
1098     console.log("Server closed");
1099 });
1100
1101 server.on("error", (err) => {
1102     console.log("Server error: ", err);
1103 });
1104
1105 server.on("clientError", (err, socket) => {
1106     console.log("Client error: ", err);
1107     socket.destroy();
1108 });
1109
1110 server.on("connection", (socket) => {
1111     console.log("New client connected");
1112 });
1113
1114 server.on("disconnect", (socket) => {
1115     console.log("Client disconnected");
1116 });
1117
1118 server.on("close", () => {
1119     console.log("Server closed");
1120 });
1121
1122 server.on("error", (err) => {
1123     console.log("Server error: ", err);
1124 });
1125
1126 server.on("clientError", (err, socket) => {
1127     console.log("Client error: ", err);
1128     socket.destroy();
1129 });
1130
1131 server.on("connection", (socket) => {
1132     console.log("New client connected");
1133 });
1134
1135 server.on("disconnect", (socket) => {
1136     console.log("Client disconnected");
1137 });
1138
1139 server.on("close", () => {
1140     console.log("Server closed");
1141 });
1142
1143 server.on("error", (err) => {
1144     console.log("Server error: ", err);
1145 });
1146
1147 server.on("clientError", (err, socket) => {
1148     console.log("Client error: ", err);
1149     socket.destroy();
1150 });
1151
1152 server.on("connection", (socket) => {
1153     console.log("New client connected");
1154 });
1155
1156 server.on("disconnect", (socket) => {
1157     console.log("Client disconnected");
1158 });
1159
1160 server.on("close", () => {
1161     console.log("Server closed");
1162 });
1163
1164 server.on("error", (err) => {
1165     console.log("Server error: ", err);
1166 });
1167
1168 server.on("clientError", (err, socket) => {
1169     console.log("Client error: ", err);
1170     socket.destroy();
1171 });
1172
1173 server.on("connection", (socket) => {
1174     console.log("New client connected");
1175 });
1176
1177 server.on("disconnect", (socket) => {
1178     console.log("Client disconnected");
1179 });
1180
1181 server.on("close", () => {
1182     console.log("Server closed");
1183 });
1184
1185 server.on("error", (err) => {
1186     console.log("Server error: ", err);
1187 });
1188
1189 server.on("clientError", (err, socket) => {
1190     console.log("Client error: ", err);
1191     socket.destroy();
1192 });
1193
1194 server.on("connection", (socket) => {
1195     console.log("New client connected");
1196 });
1197
1198 server.on("disconnect", (socket) => {
1199     console.log("Client disconnected");
1200 });
1201
1202 server.on("close", () => {
1203     console.log("Server closed");
1204 });
1205
1206 server.on("error", (err) => {
1207     console.log("Server error: ", err);
1208 });
1209
1210 server.on("clientError", (err, socket) => {
1211     console.log("Client error: ", err);
1212     socket.destroy();
1213 });
1214
1215 server.on("connection", (socket) => {
1216     console.log("New client connected");
1217 });
1218
1219 server.on("disconnect", (socket) => {
1220     console.log("Client disconnected");
1221 });
1222
1223 server.on("close", () => {
1224     console.log("Server closed");
1225 });
1226
1227 server.on("error", (err) => {
1228     console.log("Server error: ", err);
1229 });
1230
1231 server.on("clientError", (err, socket) => {
1232     console.log("Client error: ", err);
1233     socket.destroy();
1234 });
1235
1236 server.on("connection", (socket) => {
1237     console.log("New client connected");
1238 });
1239
1240 server.on("disconnect", (socket) => {
1241     console.log("Client disconnected");
1242 });
1243
1244 server.on("close", () => {
1245     console.log("Server closed");
1246 });
1247
1248 server.on("error", (err) => {
1249     console.log("Server error: ", err);
1250 });
1251
1252 server.on("clientError", (err, socket) => {
1253     console.log("Client error: ", err);
1254     socket.destroy();
1255 });
1256
1257 server.on("connection", (socket) => {
12
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
PS C:\Users\Hayden Duong\Desktop\SIIT314_729 - Software Architecture and Scalability for IOT\Module 1 - Scalability\tech_task> node socket_server.js
TCP socket server is running on port: 5000. Access via:
  - Local: 127.0.0.1:5000
  - LAN: 10.141.30.194:5000
Client 3 connected from: 10.141.7.166:56490 | sent: sub,0,9
Calculation Result: -9

Client 1 connected from: 10.141.7.166:56489 | sent: add,2,3
Calculation Result: 5

Client 2 connected from: 10.141.7.166:56488 | sent: add,2,6
Calculation Result: 8

Client 2:56488 disconnected
Client 1:56489 disconnected
Client 3:56490 disconnected
```

Picture 16 – Result from the Server-side when another team member used my “stress_test.js” to test connection to the “socket_server.js”

F/ Put Pressure on Server

```
JS socket_server.js JS stress_test.js X
JS stress_test.js > numClients
1  const { randomInt } = require("crypto");
2  const net = require("net");
3
4  const host = "192.168.4.22";
5  const port = 5000;
6  const numClients = 17000;
7
8
9  // Create a map to store client commands (key: client ID, value: command)
10 const clientCommands = new Map();
11
12 for (let i = 1; i <= numClients; i++) {
13   const client = net.createConnection(port, host, () => {
14     // Log the connection to the console
15     // Display the client ID and server address in the client-side console
16     console.log(`Client ${i} connected to server at ${host}:${port}`);
17
18     // Generate a random command for the client
19
20   });
21 }
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570

```

```

PS C:\Users\Hayden Duong\Desktop\SIIT314_729 - Software Architecture and Scalability for IoT\Module 1 - Scalability\tech_task> node stress_test.js
node:events:496
    throw er; // Unhandled 'error' event
    ^

Error: connect ECONNREFUSED 192.168.4.22:5000 - Local (undefined:undefined)
    at internalConnect (node:net:1115:16)
    at defaultTriggerAsyncIdScope (node:internal/async_hooks:464:18)
    at node:net:1355:9
    at process.processTicksAndRejections (node:internal/process/task_queues:85:11)
Emitted 'error' event on Socket instance at:
    at emitErrorNT (node:internal/stream_base_commons:178:8)
    at emitErrorCloseNT (node:internal/stream_base_commons:129:3)
    at process.processTicksAndRejections (node:internal/process/task_queues:90:21) {
  errno: -4069,
  code: 'ECONNREFUSED',
  syscall: 'connect',
  address: '192.168.4.22',
  port: 5000
}

Node.js v22.16.0

```

Picture 17 – Result from Client-side.

Week 2

A / Class Discussion Evidence

IoT Nodes

1. What capabilities does an IoT node need?

- **Sensing/Actuating:** Measures environmental data (temperature, humidity, motion) or performs actions (turning devices on/off).
- **Processing:** Basic data computation (microcontrollers, SoCs).
- **Communication:** Network connectivity (Wi-Fi, Bluetooth, LoRa, NB-IoT, etc.).
- **Power Management:** Energy efficiency (battery, solar, energy harvesting).
- **Security:** Data protection (encryption, authentication).

2. What hardware components does an IoT node need?

- **Sensors/Actuators:** Collect data or interact with the environment.
- **Microcontroller (MCU):** Processes data (e.g., ESP32, Arduino, Raspberry Pi Pico).
- **Communication Module:** Wi-Fi, BLE, Zigbee, LoRa, or Cellular (4G/5G).
- **Power Source:** Battery, solar panel, or wired connection.
- **Memory:** Temporary data storage (RAM/Flash).

3. Identify some commercially available IoT nodes.

- **Raspberry Pi (Compute Module 4):** For complex IoT applications.
- **ESP32/ESP8266:** Low-cost, supports Wi-Fi/BLE.
- **Arduino MKR Series:** Supports LoRa, NB-IoT.
- **STM32 B-L4S5I-IOT01A:** Multi-protocol IoT development board.
- **Commercial IoT Devices:** Xiaomi Mi Temperature Sensor, Philips Hue Smart Lights.

Communications

1. What do we need to consider in IoT communications?

- **Range:** Short-range (Bluetooth) vs. long-range (LoRa, Cellular).
- **Power Consumption:** Battery-powered devices need low-power protocols (NB-IoT, Zigbee).
- **Bandwidth:** Small data (sensors) vs. large data (video streaming).
- **Latency:** Real-time applications (industrial IoT) require low latency.
- **Security:** Encryption to prevent cyberattacks.

2. What IoT communication technologies exist?

- **Short-range:**
 - **Wi-Fi** (ESP32 for high-speed data).
 - **Bluetooth/BLE** (wearables, beacons).
 - **Zigbee** (smart home devices like Philips Hue).
- **Long-range (LPWAN):**
 - **LoRa** (agriculture, smart cities).
 - **NB-IoT** (low-power, wide-area coverage).

- **LTE-M** (4G/5G for mobile IoT).

3. Identify IoT nodes with different communication technologies.

- **LoRa: Dragino LoRa Node** (environmental monitoring).
- **NB-IoT: Quectel BC66** (smart city devices).
- **Zigbee: Xiaomi Aqara Sensors** (home automation).
- **Wi-Fi/BLE: ESP32 DevKit** (prototyping).

Cloud Computing for IoT

1. How does Cloud computing support IoT?

- **Data Storage & Processing:** Handles massive IoT data (Big Data).
- **Scalability:** Expands as more devices connect.
- **Real-time Analytics:** Processes live data (e.g., predictive maintenance).
- **AI/ML Integration:** Enables smart analytics (e.g., image recognition from IoT cameras).
- **Remote Management:** Controls devices via cloud platforms (AWS IoT Core, Azure IoT Hub).

2. Key differences between public, private, and hybrid cloud

Cloud Type	Definition	Pros	Cons
Public	Shared cloud services (AWS, Azure, Google Cloud).	Low cost, easy scheduling.	Less customization, security concerns.
Private	Dedicated cloud for one organization (on-premise/hosted).	High security, customizable.	Expensive, requires maintenance.
Hybrid	Combines public + private clouds	Flexible, balanced cost/security	Complex integration

B / Group Activity Evidence

Consider the following applications and answer the questions about the most appropriate technologies to use.

1. A smart home lighting system.	
What properties does your IoT node need?	<ul style="list-style-type: none">• Low power consumption.• Real-time control.• Local wireless connectivity.• Sensor (motion, light).
What microcontroller would you use?	<ul style="list-style-type: none">• ESP32 / ESP8266.<ul style="list-style-type: none">○ Cheap and common.○ Wi-Fi integrated.○ Able to handle the signal and communicate with server.
What communications standard would you use?	<ul style="list-style-type: none">• Wi-Fi.
What edge computing infrastructure would you use?	<ul style="list-style-type: none">• Local Gateway:<ul style="list-style-type: none">○ Raspberry Pi.○ Home Control Hub
Would you use cloud computing?	<ul style="list-style-type: none">• Yes & No.

2. A forest fire monitoring system.	
What properties does your IoT node need?	<ul style="list-style-type: none">• Low power (on Battery / Solar).• Long-range communication.• Environmental factor sensors.• High durability.
What microcontroller would you use?	<ul style="list-style-type: none">• STM32 / Arduino MKR WAN 1310.

What communications standard would you use?	<ul style="list-style-type: none"> • LoRa Wan – Long-range communication (up to 10Km)
What edge computing infrastructure would you use?	<ul style="list-style-type: none"> • Local LoRa Gateway.
Would you use cloud computing?	<ul style="list-style-type: none"> • Yes

3. A smart air-conditioning system for a large building.	
What properties does your IoT node need?	<ul style="list-style-type: none"> • Real-time environmental sensors (temp, CO₂, moist). • Manually controllable (Hybrid: Automate + Manually).
What microcontroller would you use?	<ul style="list-style-type: none"> • ESP32 <ul style="list-style-type: none"> ○ Wi-Fi integrated
What communications standard would you use?	<ul style="list-style-type: none"> • Wi-Fi – for within floor. • Bluetooth Low Energy (BLE) – for within a small area. • Ethernet – for between floors
What edge computing infrastructure would you use?	<ul style="list-style-type: none"> • Building Management System.
Would you use cloud computing?	<ul style="list-style-type: none"> • Yes. <ul style="list-style-type: none"> ○ Analyze power-usage. ○ Predictive maintenance.

4. A river monitoring system for the remote Australian outback.

What properties does your IoT node need?	<ul style="list-style-type: none"> • Solar-powered. • Waterproof. • Long-range communications. • Environmental factor sensors (water level, quality).
What microcontroller would you use?	<ul style="list-style-type: none"> • STM32. • Arduino MKR WAN 1310. • Adafruit Feather M0 + LoRa.
What communications standard would you use?	<ul style="list-style-type: none"> • LoRa WAN. • NB-IoT.
What edge computing infrastructure would you use?	<ul style="list-style-type: none"> • Remote Gateway. • Local LoRa Gateway
Would you use cloud computing?	<ul style="list-style-type: none"> • Yes. <ul style="list-style-type: none"> ○ Observe water-level for flood-warning.

5. A driver-less taxi system for a smart city.	
What properties does your IoT node need?	<ul style="list-style-type: none"> • High-speed processing. • Real-time sensors (camera, GPS). • Reliable & fast networking.
What microcontroller would you use?	<ul style="list-style-type: none"> • Raspberry Pi 5. <ul style="list-style-type: none"> ○ Support Cellular 5G
What communications standard would you use?	<ul style="list-style-type: none"> • 5G.
What edge computing infrastructure would you use?	<ul style="list-style-type: none"> • On-board Edge Server – within the vehicle. • Roadside Edge Server.

Would you use cloud computing?	<ul style="list-style-type: none"> • Yes. <ul style="list-style-type: none"> ○ Fleet-management.
--------------------------------	---

6. A system for automatic robotic maintenance on Mars.	
What properties does your IoT node need?	<ul style="list-style-type: none"> • Radiation tolerance and high durability. • Autonomy. • Strong and Reliable Communication.
What microcontroller would you use?	<ul style="list-style-type: none"> • RAD-tolerant MCU. <ul style="list-style-type: none"> ○ TI TMS570
What communications standard would you use?	<ul style="list-style-type: none"> • UHF / VHF. • X-band. • Delay Tolerant Networking (DTN).
What edge computing infrastructure would you use?	<ul style="list-style-type: none"> • Local AI. • Self-Decision-making system.
Would you use cloud computing?	<ul style="list-style-type: none"> • No. <ul style="list-style-type: none"> ○ Due to high latency between Earth and Mar.

C / Technical Task Evidence

IoT Weather Warning System - Technical Documentation

This document details the specific modifications and additions made to the IoT Weather Warning System's components, explaining the reason behind each change.

Fire Data Node (fire_node.js – Code Snippet 1) - New Implementation

The fire_node.js was introduced as a new component to fetch real-time fire danger ratings from the Country Fire Authority (CFA) Victoria and send them to the central Weather Service.

Key Implementation Details and Reasoning:

- **Fetching RSS Feed:** We used the rss-parser library to retrieve the XML RSS feed from https://www.cfa.vic.gov.au/cfa/rssfeed/tfbfdrforecast_rss.xml.
 - **Reasoning:** rss-parser simplifies the process of parsing RSS/Atom feeds into manageable JavaScript objects, allowing us to easily access feed items.
- **Handling Status code 406 Error:** Initially, requests to the CFA RSS feed resulted in a Status code 406 (Not Acceptable) error. This was resolved by configuring rss-parser to send standard HTTP headers.
 - **Explanation:** Many web servers, including those hosting RSS feeds, inspect HTTP request headers (like User-Agent and Accept) to determine if the client is a legitimate browser or an automated script. By sending common browser-like headers, we made our requests appear valid to the CFA server.
- **Correcting latestItem.description to latestItem.content:** After resolving the 406 error, a new issue arose where cheerio.load() expected a string, but latestItem.description was undefined. Debugging revealed that rss-parser mapped the RSS item's <description> XML tag to the content property of the parsed JavaScript object, not description.
 - **Explanation:** rss-parser has conventions for how it maps XML elements to JavaScript object properties. In this case, the main HTML content within the <description> tag was made available via the content property.
- **Parsing HTML with cheerio:** The extracted content (which is an HTML string) contains the fire danger ratings. cheerio was used to navigate this HTML structure.
 - **Reasoning:** cheerio provides a jQuery-like syntax for parsing and manipulating HTML, making it easy to find specific elements (like <p> tags) and extract their text or HTML content.
 - **Process:** We specifically looked for the <p> tag containing "Fire Danger Ratings", then extracted data from the *next* <p> tag, splitting it by
 tags to get individual area ratings.

- **Data Transmission to Weather Service:** The extracted fire warning levels are formatted as a JSON object (e.g., {"Central": "NO RATING", "Mallee": "LOW"}) and sent via TCP socket to the Weather Service using the command fire,{"JSON_DATA"}.
 - **Reasoning:** JSON is a lightweight and human-readable format for transmitting structured data, making it suitable for this purpose.
 - **Periodic Updates:** The fetchFireWarning function is called every 5 minutes using setInterval.
 - **Reasoning:** Fetching too frequently (e.g., every few seconds) can lead to IP blocking or rate-limiting by the external RSS feed provider. A 5-minute interval is generally more polite and sustainable for public data sources.
-

Weather Service Node (weather_service.js – Code Snippet 2) - Modifications

The central weather_service.js was modified to correctly receive and process the new fire data from fire_node.js and to handle request commands for localized warnings.

Key Modifications and Reasoning:

- **Correct JSON Payload Parsing:** The primary issue here was that the data.toString().split(',') method incorrectly split the incoming JSON string from the fire_node.js.
 - **Explanation:** When fire_node.js sends fire,{"Central":"NO RATING", ...}, the JSON string itself contains commas. A simple split(',') would break this JSON into multiple pieces (e.g., command[1] would be {"Central":"NO RATING", command[2] would be "East Gippsland":"NO RATING"}). This led to JSON.parse() receiving an incomplete and invalid string.
 - **Solution:** We implemented a more robust parsing method using indexOf(',') and substring() to ensure only the *first* comma separates the command name from the entire JSON payload.
- **Handling fire command:** A new case "fire": was added to the switch statement to process the incoming fire warning data.
 - **Explanation:** This case is responsible for parsing the rawCommandValue (which is now the full JSON string) into the fireWarningLevels object using JSON.parse(). This object then stores the fire danger ratings for each region.

- **Handling request command with Localization:** The request command was updated to accept an areaToRequest parameter.
 - **Explanation:** Previously, request might have been generic. Now, rawCommandValue after the split for a request command directly contains the area name (e.g., "Central"). The service uses this area name to look up the specific fire warning from the fireWarningLevels object and includes it in the response.
-

Warning Request Node (warning_request.js – Code Snippet 3) - Modifications

The warning_request.js client was modified to enable requesting warnings for specific localized areas.

Key Modifications and Reasoning:

- **Dynamic Area Request:** A new const areaToRequest variable was introduced.
 - **Explanation:** This allows the client to specify which Victorian region it wants to query (e.g., "Central"). This makes the request node more flexible and capable of simulating requests from different geographical locations.
- **Purpose:** This modification ensures that the system can demonstrate the delivery of **localized warnings**, which was a key requirement.

Code of fire_node.js, modified weather_service.js, modified warning_request.js

```
// fire_node.js

// This code implements a TCP socket client that connects to a weather service
// and fetches fire warning levels from an RSS feed, sending updates to the server.

const net = require("net");
const Parser = require("rss-parser");
const cheerio = require("cheerio");

const host = "127.0.0.1";
const port = 6000;
const CFA_RSS_FEED_URL = "https://www.cfa.vic.gov.au/cfa/rssfeed/tfbfdrforecast_rss.xml";
```

```

// Create a TCP client
const client = new net.Socket();

// Initialize variables to store fire warning levels
let fireWarningLevels = {};

// Function to fetch fire warning levels from the RSS feed
async function fetchFireWarning() {
  try {
    // Use rss-parser to fetch and parse the RSS feed
    // Ensure the parser is configured correctly
    // to handle the specific structure of the CFA RSS feed
    // Note: The CFA RSS feed may have specific headers or content types
    // that need to be handled, so we use a custom parser configuration
    let parser = new Parser({
      headers: {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36',
        'Accept': 'application/xml, text/xml, */*'
      }
    });

    let feed = await parser.parseURL(CFA_RSS_FEED_URL);

    // Check if the feed has items
    // If items are present, process the first item (usually today)
    // If no items are found, set a default warning level
    if (feed.items.length > 0) {
      const latestItem = feed.items[0];

      // Use cheerio to parse the content of the latest item
      // and extract the fire warning levels
      // This assumes the content is HTML and contains the relevant information
      const $ = cheerio.load(latestItem.content);
    }
  }
}

```



```

// Initialize fireWarningLevels to an empty object
fireWarningLevels = {};

// Find the paragraph that contains the "Fire Danger Ratings"
const fireDangerRatingsHeaderP = $('p').filter(function() {
    return $(this).text().trim() === 'Fire Danger Ratings';
});

// If the header is found (or there is a sentence called "Fire Danger Ratings"), proceed to extract the ratings
if (fireDangerRatingsHeaderP.length > 0) {

    // Find the next paragraph that contains the actual ratings
    // This assumes the ratings are in the next paragraph after "Fire Danger Ratings"
    const actualRatingsP = fireDangerRatingsHeaderP.next('p');
    if (actualRatingsP.length > 0) {

        // Split the ratings by line breaks and process each line
        // to extract area names and their corresponding ratings
        // e.g., "Central: NO RATING<br>East Gippsland: NO RATING<br>Mallee: NO RATING<br>North Central: NO
        RATING<br>..."
        const rawRatings = actualRatingsP.html();

        // Split the raw ratings by line breaks and store them in an array
        // This assumes the ratings are separated by <br> tags
        // Outcome: ratingLines = ["Central: NO RATING", "East Gippsland: NO RATING", ...]
        const ratingLines = rawRatings.split('<br>');

        // Process each element of ratingLines to extract area names and ratings
        // and store them in the fireWarningLevels object
        ratingLines.forEach(element => {
            // Trim whitespace and match the pattern "Area: Rating"
            // e.g., "Central: NO RATING"
            // Outcome: fireWarningLevels = { "Central": "NO RATING", "East Gippsland": "NO RATING", ... }
            const trimmedElement = element.trim();

```

```

const match = trimmedElement.match(/^(.+?):\s*(.+)$/);

// If the match is successful, extract area name and rating
// and store them in the fireWarningLevels object
// eg., match = ["Central: NO RATING", "Central", "NO RATING"] where:
// match[0] = "Central: NO RATING" (full match) comes from ^
// match[1] = "Central" (area name) comes from (.+?)
// match[2] = "NO RATING" (rating) comes from \s*(.+)$

if (match && match.length === 3) {
    let areaName = match[1].trim();
    let rating = match[2].trim();

    // fireWarningLevels["Central"] = "NO RATING"
    fireWarningLevels[areaName] = rating;
}

});

} else {
    console.warn("Warning: Could not find the actual fire ratings paragraph.");
    fireWarningLevels = { "ALL": "NO RATING - PARAGRAPH_NOT_FOUND" };
}

} else {
    console.warn("Warning: Could not find 'Fire Danger Ratings' header.");
    fireWarningLevels = { "ALL": "NO RATING - HEADER_NOT_FOUND" };
}

console.log("Fire warning levels fetched successfully:", fireWarningLevels);

} else {
    console.log("No items found in the RSS feed.");
    fireWarningLevels = { "ALL": "NO RATING - NO_ITEMS" };
}

} catch (error) {
    console.error("Error fetching fire warning levels:", error.message);
}

```

```

    fireWarningLevels = { "ALL": "ERROR_FETCHING" };
}

// Send the fire warning levels to the server
// Check if the client is connected before sending data
// If the client is not connected, attempt to reconnect and send data
if (client.readyState === 'open') {
    client.write(`fire,${JSON.stringify(fireWarningLevels)}`);
    console.log(`Sent: fire,${JSON.stringify(fireWarningLevels)}`);
} else {
    console.log("Client is not connected, attempting to reconnect and send data.");

    // Attempt to reconnect if the client is not connected
    // This is a simple retry mechanism to ensure the client can send data
    // after reconnecting to the server
    if (client.connecting === false && client.destroyed === false) {
        client.connect(port, host, () => {
            console.log("Reconnected to Weather Service");
            client.write(`fire,${JSON.stringify(fireWarningLevels)}`);
            console.log(`Sent after reconnect: fire,${JSON.stringify(fireWarningLevels)}`);
        });
    } else {
        console.log("Client is already connecting or destroyed. Will try again next interval.");
    }
}

}

// Connect to the weather service
client.connect(port, host, () => {
    console.log("Connected to Weather Service");

    fetchFireWarning();

    setInterval(fetchFireWarning, 300000);

```

```

});

// Handle incoming data from the server
client.on("data", (data) => {
  console.log(`Received from Weather Service: ${data.toString()}`);
});

// Handle socket error event
client.on("error", (error) => {
  console.error(`Client Error: ${error.message}`);
});

// Handle socket end event
client.on("close", () => {
  console.log("Connection closed");
});

```

Code Snippet 1 – fire_node.js

```

// weather_service.js
// This code implements a TCP socket server that listens for weather data updates
// and responds to requests for weather conditions.
const net = require("net");
const port = 6000;

// Initialize variables to store weather data
let temp;
let wind;
let rain;
let fireWarningLevels = {};

// Create a TCP server
const server = net.createServer((socket) => {
  console.log("Client connected");

  // Set the encoding for the socket

```

```
socket.on("data", (data) => {  
    const strData = data.toString().trim();  
    console.log(`Received: ${strData}`);  
  
    const firstCommaIndex = strData.indexOf(",");  
  
    if (firstCommaIndex === -1) {  
        console.error("Invalid command format: No comma found.");  
        socket.write("error");  
        return;  
    }  
  
    const name = strData.substring(0, firstCommaIndex);  
    const rawCommandValue = strData.substring(firstCommaIndex + 1);  
  
    // Process the command based on the name  
    switch (name) {  
        case "temp":  
            temp = parseFloat(rawCommandValue);  
            console.log(name + " : " + temp);  
            result = "ok";  
            break;  
  
        case "rain":  
            rain = parseFloat(rawCommandValue);  
            console.log(name + " : " + rain);  
            result = "ok";  
            break;  
  
        case "wind":  
            wind = parseFloat(rawCommandValue);  
            console.log(name + " : " + wind);  
            result = "ok";  
            break;
```

```
case "fire":

    try {

        fireWarningLevels = JSON.parse(rawCommandValue);

        console.log("Received Fire Warning Levels:", fireWarningLevels);

        result = "ok";

    } catch (error) {

        console.error("Error parsing fire warning levels:", error);

        result = "error";

    }

    break;

case "request":

    const requestedArea = rawCommandValue;

    console.log(`Request for weather conditions in area: ${requestedArea}`);

    const currentFireWarning = fireWarningLevels[requestedArea] || "NO DATA";

    // Respond with the current weather conditions
    if(temp > 20 && rain < 50 && wind > 30){

        result = `Weather Warning. Fire Warning Level: ${currentFireWarning}`;

    }

    else {

        result = `Everything is fine. Fire Warning Level: ${currentFireWarning}`;

    }

    break;

}

// Send the result back to the client

socket.write(result.toString());

});

// Handle socket end event
socket.on("end", () => {

    console.log("Client disconnected");
```

```

});

// Handle socket error event
socket.on("error", (error) => {
    console.log(`Socket Error: ${error.message}`);
});
});

// Handle server errors
server.on("error", (error) => {
    console.log(`Server Error: ${error.message}`);
});

// Start the server and listen on the specified port
server.listen(port, () => {
    console.log(`TCP socket server is running on port: ${port}`);
});

```

Code Snippet 2 – Modified weather_service.js to complement with fire_node.js

```

// warning_request.js
// This code implements a TCP socket client that sends a request for weather conditions
// and listens for responses from the server.
const net = require("net");

const host = "127.0.0.1";
const port = 6000;

// Specify the area for which to request weather conditions
// List of areas can be defined based on the fire warning levels:
// Central, East Gippsland, Mallee, North Central, North East, Northern Country, South West, West and South Gippsland,
// Wimmera
const areaToRequest = "Central"; // Specify the area for which to request weather conditions

// Create a TCP client that connects to the server

```

```
const client = net.createConnection(port, host, () => {
  console.log("Connected to Weather Service");

  // Set the encoding for the client
  setInterval(() => {
    client.write(`request,${areaToRequest}`);
    console.log(`Sent request for weather conditions in area: request,${areaToRequest}`);
  }, 2000); // Interval in milliseconds (2000ms = 2 seconds)
});

// Handle incoming data from the server
client.on("data", (data) => {
  console.log(`Received: ${data.toString()}`);
//   process.exit(0);
});

// Handle socket error event
client.on("error", (error) => {
  console.log(`Error: ${error.message}`);
});

// Handle socket end event
client.on("close", () => {
  console.log("Connection closed");
});
```

Code Snippet 3 – Modified warning_request.js

Photo:


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

PS C:\Users\Hayden Duong\Desktop\SIT314_729 - Software Architecture and Scalability for IOT\Module 1 - Scalability\tech_task_week_2> node weather_service.js
TCP socket server is running on port: 6000
Client connected
temp : 31
Client connected
rain : 198
Client connected
wind : 1
Client connected
Received Fire Warning Levels: {
  Central: 'NO RATING',
  'East Gippsland': 'NO RATING',
  Mallee: 'NO RATING',
  'North Central': 'NO RATING',
  'North East': 'NO RATING',
  'Northern Country': 'NO RATING',
  'South West': 'NO RATING',
  'West and South Gippsland': 'NO RATING',
  Wimmera: 'NO RATING'
}
Client connected
Request for weather conditions in area: Central
█
```

Picture 1 – weather_service.js server received data from temperature_node, rain_node, wind_node, fire_node, and from warning_request

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

PS C:\Users\Hayden Duong\Desktop\SIT314_729 - Software Architecture and Scalability for IOT\Module 1 - Scalability\tech_task_week_2> node temperature_node.js
Connected
Received: ok
█
```

Picture 2 – temperature_node is successfully connected to weather_service server.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

PS C:\Users\Hayden Duong\Desktop\SIT314_729 - Software Architecture and Scalability for IOT\Module 1 - Scalability\tech_task_week_2> node rain_node.js
Connected
Received: ok
█
```

Picture 3 – rain_node is successfully connected to weather_service server.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

PS C:\Users\Hayden Duong\Desktop\SIT314_729 - Software Architecture and Scalability for IOT\Module 1 - Scalability\tech_task_week_2> node wind_node.js
Connected
Received: ok
█
```

Picture 4 - wind_node is successfully connected to weather_service server.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

PS C:\Users\Hayden Duong\Desktop\SIT314_729 - Software Architecture and Scalability for IOT\Module 1 - Scalability\tech_task_week_2> node fire_node.js
Connected to Weather Service
Fire warning levels fetched successfully: {
  Central: 'NO RATING',
  'East Gippsland': 'NO RATING',
  Mallee: 'NO RATING',
  'North Central': 'NO RATING',
  'North East': 'NO RATING',
  'Northern Country': 'NO RATING',
  'South West': 'NO RATING',
  'West and South Gippsland': 'NO RATING',
  Wimmera: 'NO RATING'
}
Sent: fire,{"Central":"NO RATING","East Gippsland":"NO RATING","Mallee":"NO RATING","North Central":"NO RATING","North East":"NO RATING","Northern Country":"NO RATING","South West":"NO RATING","West and South Gippsland":"NO RATING","Wimmera":"NO RATING"}
Received from Weather Service: ok
█
```

Picture 5 – fire_node is successfully connected to weather_service server.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
PS C:\Users\Hayden Duong\Desktop\SIT314_729 - Software Architecture and Scalability for IOT\Module 1 - Scalability\tech_task_week_2> node warning_request.js
Connected to Weather Service
Sent request for weather conditions in area: request,Central
Received: Everything is fine. Fire Warning Level: NO RATING
```

Picture 6 – warning_request is successfully connected to weather_service server and received warning result.

GitHub Link for code of both week 1 and week 2 technical task:

https://github.com/HaydenDuong/SIT314_729---Software-Architecture-and-Scalability-for-IOT/tree/main/Module%201%20-%20Scalability

Module Summary:

From this module, I learned about the concept of IoT – what components make up of it, how there are interconnected / communicate with one another (from receiving the signals to response accordingly to those signals), the architecture (N-Tier) and different kinds of nodes that could present in a common IoT structure that we are use daily. These will serve as a strong foundation for me to prepare for the upcoming materials and knowledge in the following weeks to build up.