

# TABA Report

---

Miguel Angel Vinas  
x22116133

## OVERVIEW

*This document is submitted as a submission of the TABA project of the Distributed Systems module on the Higher Diploma in Computing specializing in Software Development delivered by Yasantha Samarawickrama.*

1.

By using the MQTT protocol implement using JavaScript, the Publisher – Subscriber parts of the following application. Your application simulates a smart agriculture environment, where sensors emit different data about the farm.

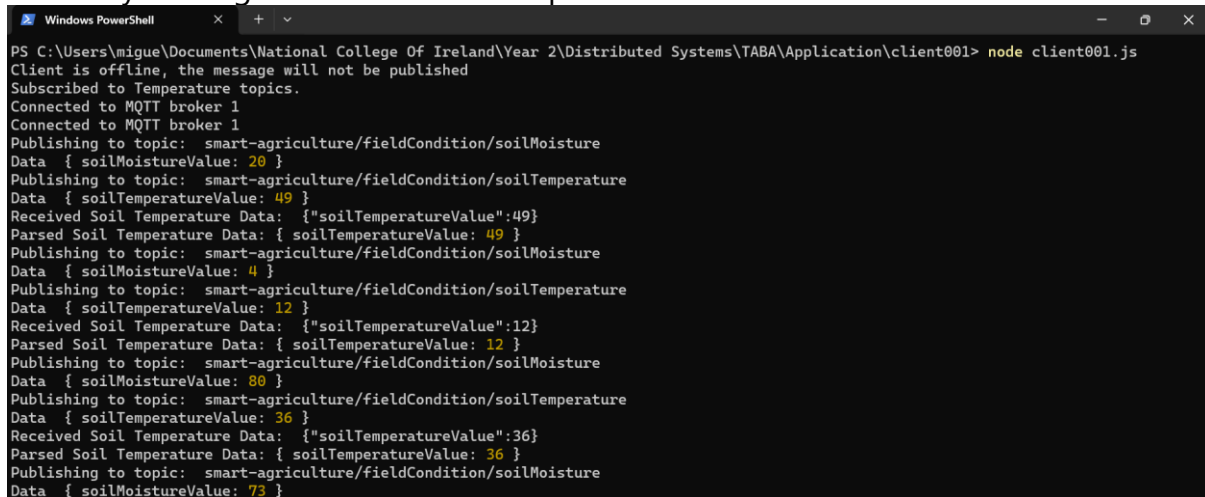
There are two types of clients (programs):

1. One client (program) that publishes messages every second about the field conditions, such as soil/moisture (e.g., 40%) and soil/temperature (e.g., 25°C),
2. Another client (program) that publishes messages about the equipment every half second, like sprinkler/harvester equipment status (e.g., ON) and sprinkler/harvester equipment location (e.g., GPS coordinates).

a. Implement two publishers, one for each client, and demonstrate the sending of messages on the respective topics and subtopics. Use a loop that periodically emits values/readings either randomly or from a list.

You should also develop and run different subscribers (hint - copy and change the class name and subscribe method in each case) that listens for messages on the following topics and subtopics

b. Strictly messages related to soil temperature.

A screenshot of a Windows PowerShell terminal window. The title bar reads 'Windows PowerShell'. The terminal shows the execution of a Node.js script 'client001.js'. The output includes status messages like 'Client is offline, the message will not be published', 'Subscribed to Temperature topics.', and 'Connected to MQTT broker 1'. It then shows a series of publish and receive operations for topics 'smart-agriculture/fieldCondition/soilMoisture' and 'smart-agriculture/fieldCondition/soilTemperature'. The received data is shown as JSON objects with 'soilTemperatureValue' keys and values like 49, 12, 36, and 73.

```
PS C:\Users\miguel\Documents\National College Of Ireland\Year 2\Distributed Systems\TABA\Application\client001> node client001.js
Client is offline, the message will not be published
Subscribed to Temperature topics.
Connected to MQTT broker 1
Connected to MQTT broker 1
Publishing to topic: smart-agriculture/fieldCondition/soilMoisture
Data { soilMoistureValue: 20 }
Publishing to topic: smart-agriculture/fieldCondition/soilTemperature
Data { soilTemperatureValue: 49 }
Received Soil Temperature Data: {"soilTemperatureValue":49}
Parsed Soil Temperature Data: { soilTemperatureValue: 49 }
Publishing to topic: smart-agriculture/fieldCondition/soilMoisture
Data { soilMoistureValue: 4 }
Publishing to topic: smart-agriculture/fieldCondition/soilTemperature
Data { soilTemperatureValue: 12 }
Received Soil Temperature Data: {"soilTemperatureValue":12}
Parsed Soil Temperature Data: { soilTemperatureValue: 12 }
Publishing to topic: smart-agriculture/fieldCondition/soilMoisture
Data { soilMoistureValue: 80 }
Publishing to topic: smart-agriculture/fieldCondition/soilTemperature
Data { soilTemperatureValue: 36 }
Received Soil Temperature Data: {"soilTemperatureValue":36}
Parsed Soil Temperature Data: { soilTemperatureValue: 36 }
Publishing to topic: smart-agriculture/fieldCondition/soilMoisture
Data { soilMoistureValue: 73 }
```

Figure 1 - Messages being published and received for the Soil Temperature through Windows PowerShell.

c. Any messages that are related to the farm field (including its subtopics).

```
Windows PowerShell
PS C:\Users\miguel\Documents\National College Of Ireland\Year 2\Distributed Systems\TAB\Application\client001> node client001.js
Client is offline, the message will not be published
Subscribed to Temperature topics.
Connected to MQTT broker 1
Connected to MQTT broker 1
Publishing to topic: smart-agriculture/fieldCondition/soilMoisture
Data { soilMoistureValue: 20 }
Publishing to topic: smart-agriculture/fieldCondition/soilTemperature
Data { soilTemperatureValue: 49 }
Received Soil Temperature Data: {"soilTemperatureValue":49}
Parsed Soil Temperature Data: { soilTemperatureValue: 49 }
Publishing to topic: smart-agriculture/fieldCondition/soilMoisture
Data { soilMoistureValue: 4 }
Publishing to topic: smart-agriculture/fieldCondition/soilTemperature
Data { soilTemperatureValue: 12 }
Received Soil Temperature Data: {"soilTemperatureValue":12}
Parsed Soil Temperature Data: { soilTemperatureValue: 12 }
Publishing to topic: smart-agriculture/fieldCondition/soilMoisture
Data { soilMoistureValue: 80 }
Publishing to topic: smart-agriculture/fieldCondition/soilTemperature
Data { soilTemperatureValue: 36 }
Received Soil Temperature Data: {"soilTemperatureValue":36}
Parsed Soil Temperature Data: { soilTemperatureValue: 36 }
Publishing to topic: smart-agriculture/fieldCondition/soilMoisture
Data { soilMoistureValue: 73 }
```

Figure 2 - Messages being published and received for the farm field.

d. Messages that are related to the equipment for both sprinkler and harvester.

```
Windows PowerShell
PS C:\Users\miguel\Documents\National College Of Ireland\Year 2\Distributed Systems\TAB\Application\client002> node client002.js
Client is offline, the message will not be published
Connected to MQTT broker 2
Subscribed to equipment topics.
Connected to MQTT broker 2
Publishing to topic: smart-agriculture/equipmentStatus/sprinkler
Data: {
  equipmentSprinklerStatus: 'ON',
  equipmentSprinklerLocation: { latitude: 0.5204067382412769, longitude: 47.129001513366234 }
}
Publishing to topic: smart-agriculture/equipmentStatus/harvester
Data: {
  equipmentHarvesterStatus: 'OFF',
  equipmentHarvesterLocation: { latitude: 88.17216938267256, longitude: 19.4463276714896 }
}
Received Equipment Data for smart-agriculture/equipmentStatus/sprinkler: {"equipmentStatus":{"latitude":0.5204067382412769,"longitude":47.129001513366234}}
Parsed Equipment Data for smart-agriculture/equipmentStatus/sprinkler: {
  equipmentStatus: { latitude: 0.5204067382412769, longitude: 47.129001513366234 }
}
Received Equipment Data for smart-agriculture/equipmentStatus/harvester: {"equipmentStatus":"OFF","equipmentLocation":{"latitude":88.17216938267256,"longitude":19.4463276714896}}
Parsed Equipment Data for smart-agriculture/equipmentStatus/harvester: {
  equipmentStatus: 'OFF',
  equipmentLocation: { latitude: 88.17216938267256, longitude: 19.4463276714896 }
}
```

Figure 3 – Messages being published and received for the sprinkler and the harvester

e. Finally, show and explain how you can facilitate disconnected clients.

```
setupListeners()
{
  //We are logging a message when our client successfully connects to the broker.
  this.client.on ('connect', () =>
  {
    console.log ('Connected to MQTT broker 2');
    //We are setting the flag to ONLINE when we connect.
    this.online = true;
  });
  //In the same way, when the client is offline we are handling the error with a message.
  //And we are setting an exponential timeout to avoid overwhelming the server when reconnecting.
  this.client.on ('offline', () =>
  {
    console.log ('Client is offline. Reconnecting...');
    //We are setting the flag to go OFFLINE when we can't connect.
    this.online = false;
    setTimeout(() =>
    {
      this.client.reconnect();
    }, reconnectTimeout);
    reconnectTimeout *= 2;
  });
}
publish (topic, message)
{
  if (this.online)
  {
    //And we are telling the system to only publish if the client is online.
    this.client.publish (topic, message);
  }
  else
  {
    //Otherwise we are going to tell the user that the client is offline.
    console.log ('Client is offline, the message will not be published');
  }
}
```

Figure 4 – Error handling of clients being disconnected / offline

2.

Conceptually demonstrate a simple client-server application/scenario with an implementation technology of your choice. Your solution should mention and depict the relations and interactions among the following components: client-part, server-part, middleware and registry. In this scenario, what is the use of middleware and registry?

Explain using any technology of your choice.

My application is a Smart Zebra Cross system that uses LED lights and an activation radius to warn drivers that a pedestrian is crossing. A user of the system can either activate or deactivate the LEDs from the zebra cross, look up in real time different data and location of the zebra cross and download logs.

I developed and implemented this application in the CA for Distributed Systems.

My scenario is a smart zebra cross that is malfunctioning, the operator / user has to deactivate it, download the logs and look up for what is wrong with the zebra cross so it can be repaired.

We are going to break down the application in four components.

*1) Client.*

The Clients of the application are the zebra cross hardware and the user's web interface.

The zebra cross hardware acts as a sensor, allowing users to activate and deactivate the LED lights from a specific zebra cross.

The user's web interface provides real time monitoring, activation / deactivation of the LED lights, accurate location of the zebra cross and log access.

*2) Server.*

The server manages the communication between the zebra crosses and the users.

It handles requests from zebra crosses, updates real time information and provides the logs to the users.

*3) Middleware (MQTT Broker).*

The MQTT broker (HiveMQ - <https://www.hivemq.com/mqtt/public-mqtt-broker/>) acts as middleware.

The zebra crosses (which are the publishers) communicate with the server (subscriber) using MQTT topics.

The user's web interface (another subscriber) receives real time updates and logs from the server through MQTT.

*4) Registry.*

In this particular use case the registry is keeping track of the zebra crosses that are registered in the system and their status (whether they are activated or deactivated).

The registry helps the server route the messages to the correct zebra cross and provides the user's web interface with the list of the available zebra crosses in the map.

## Interactions

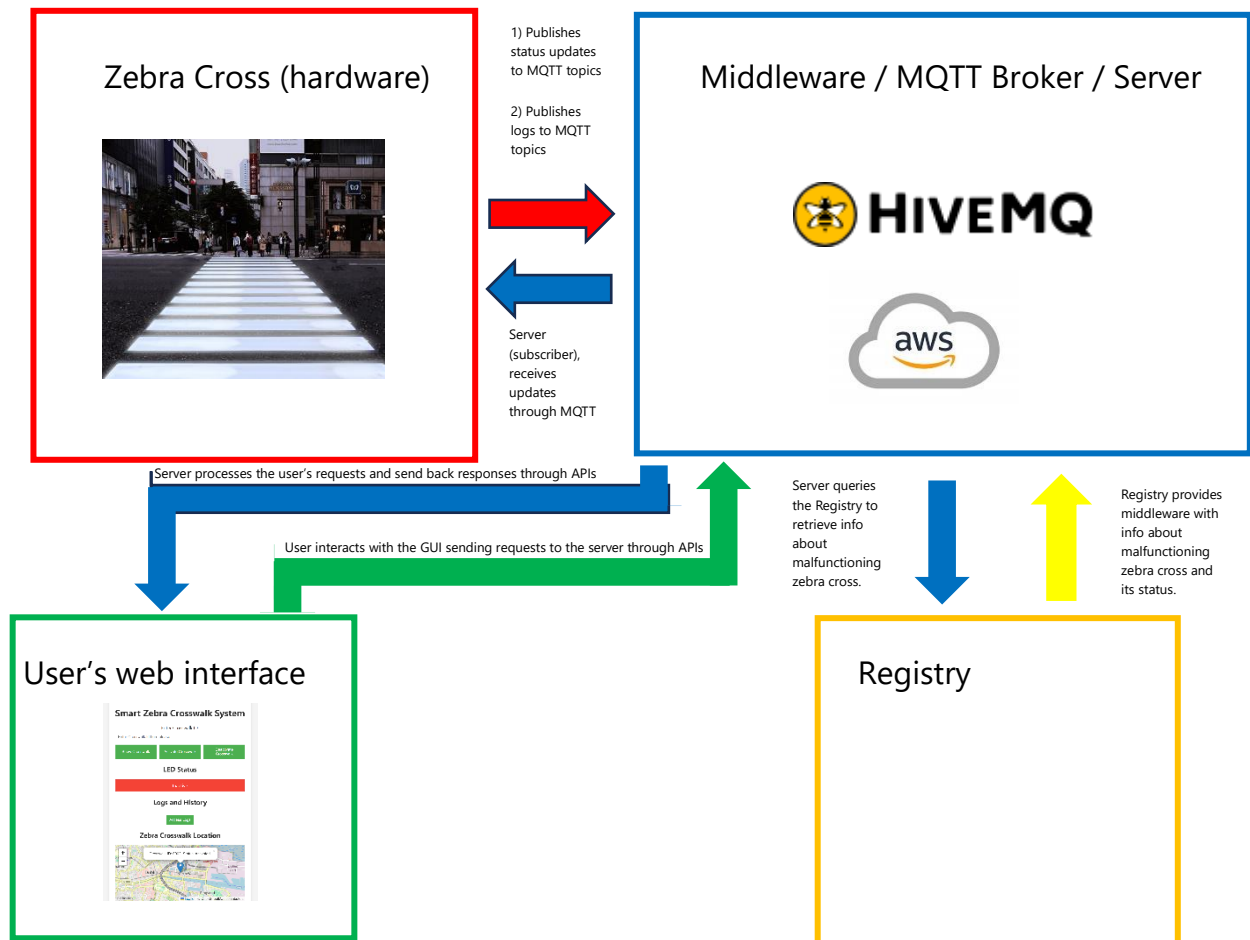


Figure 5 – Diagram of the interactions between components of the application.

3.

Discuss the challenges and advantages of adopting a microservices architecture in distributed systems. How does the use of microservices impact issues such as data consistency, fault tolerance, and system complexity?

When we create a piece of software we are following different approaches and best practices.

One of the most popular architectural patterns is the Microservices architecture, which consists in a collection of small and autonomous services that should implement a single business capability.

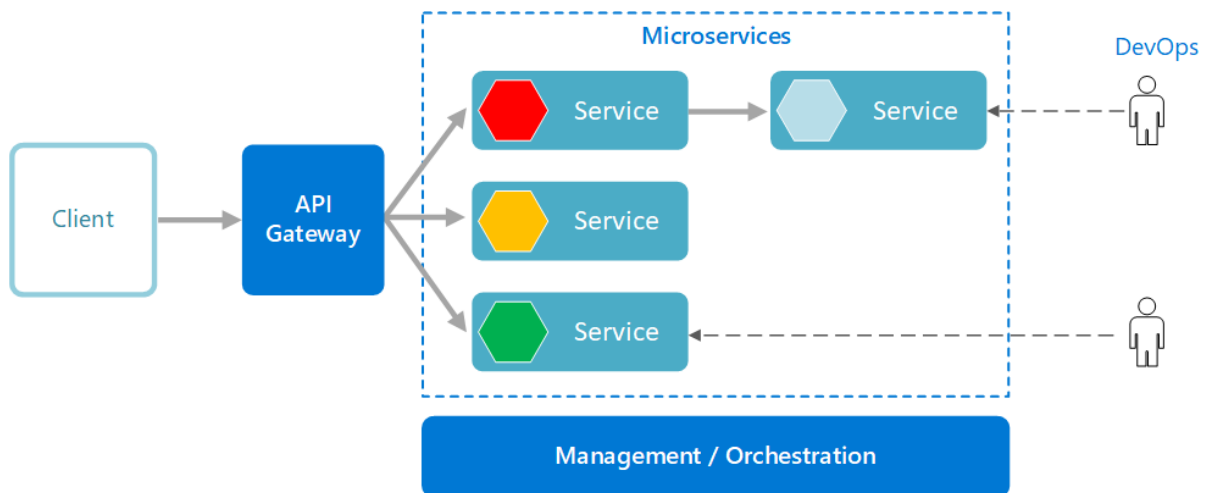


Figure 6 – Diagram of microservices - [Microsoft](#)

#### Advantages:

- 1) Scalability: Microservices allow for independent scaling of different services based on their individual needs.
- 2) Independent Deployment: Microservices can be deployed independently, facilitating continuous integration and continuous delivery. We can update a service without having to redeploy the entire piece of software.
- 3) Small teams: Microservices should be small enough that a single team should be able to build, test and deploy the code.
- 4) Technology Diversity: Microservices can use different technologies to achieve their purpose.
- 5) Fault Tolerance: Microservices enhance fault tolerance as failures in one of the services do not necessarily affect others. However, designing for fault tolerance must be included at the architecture level.

#### Challenges:

- 1) Data Consistency: Because different services might manage their data independently, maintaining consistency across them can be challenging and complex.
- 2) System Complexity: Even though microservices can simplify deployment, they introduce a new layer of complexity when managing communication and deployment.

#### 4.

Discuss mitigation strategies for the following Fallacies of Distributed Computing.

##### a. The Network is Reliable.

Networks are complex, dynamic and unpredictable, due to this complexity, networks are unreliable.

We have to accept and treat failure as a matter of course but we should design a system that is able to mitigate these failures that will occur and continue working as intended.

We need to engineer the system to be fault – tolerant and highly redundant.  
We also need to employ retry mechanisms for failed network operations to allow the system to recover.  
And we should use health checks and monitoring to proactively detect and react to network failures.

*b. Latency is zero.*

Latency is the speed at which data travels from A to B.

We should never assume that there will be no delay or zero latency between the data being sent and the data being received.

In order to minimize the delay we can bring data closer to clients. If we are building a cloud – based system, we should choose the availability zones carefully, ensuring they are in proximity to our clients and route the traffic accordingly.

When designing the systems, we should design them to be asynchronous (if possible), that will allow tasks to proceed independently without waiting for immediate responses.

*c. Bandwidth is infinite.*

Bandwidth means how much data can be transferred from one place to another in a certain timeframe.

To solve the problem of the bandwidth not being infinite we could prioritize the transfer of critical data and ensure that the bandwidth is used efficiently.  
We could also optimize the data transfer and implement data compression techniques to reduce the amount of data sent.

5.

Explain the different styles of RPC methods in gRPC. Explain using proto files and provide practical applications of each style.

gRPC is an open source framework developed by Google that enables efficient communication between services and allows developers to build distributed systems easily.

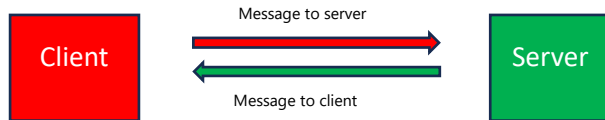
A standout feature of gRPC is its use of Protocol Buffers (protobuf defined in proto files), which are more compact and faster to parse than JSON or XML, resulting in smaller payloads and quicker transmission times.

gRPC uses HTTP/2 to transmit data between clients and servers and uses Protocol Buffers to serialize this data.

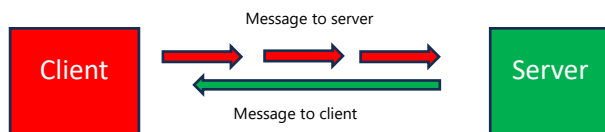


gRPC supports different communication types, including:

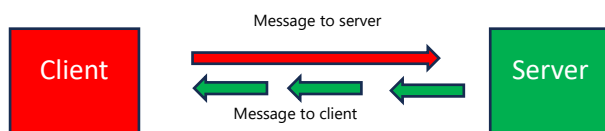
- UnaryRPC: It is a request – response RPC. The client sends a request to the server and the server sends a response to the client.



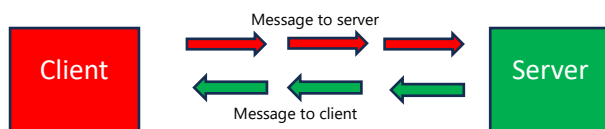
- Server Streaming: The server streaming allows servers to send multiple messages to the client for a single request in a streaming.



- Client Streaming: The client streaming allows clients to send multiple messages to the server in a streaming fashion.



- Bidirectional Streaming: Both, the client and the server can send streams of messages to each other asynchronously.



An example of a proto file is the following pedestrian\_detection.proto file implemented in the zebra crosswalk CA for Distributed Systems.

```

1  /*
2  Title: pedestrian_detection.proto
3  Author: Miguel Angel Vinas
4  Date: 1st January, 2024
5  Purpose: Distribution's Systems CA for National College Of Ireland
6  */
7
8  // We are defining the syntax (proto3 version) of the protocol buffer file
9  syntax = "proto3";
10
11 // We are declaring the package for the led_crosswalk service
12 package pedestrian_detection;
13
14 //And we are defining the PedestrianDetectionService, which contains the RPC method
15 service PedestrianDetectionService
16 {
17     rpc DetectPedestrian(DetectPedestrianRequest) returns (DetectPedestrianResponse);
18 }
19
20 // We are defining the messages for the different requests
21 message DetectPedestrianRequest {}
22
23 message DetectPedestrianResponse
24 {
25     bool presence = 1;
26 }
27
28

```

Figure 7 – Example of proto file

## 6.

### References.

HiveMQ – MQTT Essentials. [Online] <https://www.hivemq.com/mqtt/>

Medium - How I built a Client – Server desktop app. Talaver, Oleg. 27<sup>th</sup> June 2020. [Online] <https://medium.com/@vidzhel/how-i-built-a-client-server-desktop-app-50362478ccff>

Geeks For Geeks – Client – Server Mode. 2<sup>nd</sup> December 2022. [Online] <https://www.geeksforgeeks.org/client-server-model/>

Amazon - What is Middleware?. [Online] <https://aws.amazon.com/what-is/middleware/>

HiveMQ. [Online] <https://www.hivemq.com/mqtt/public-mqtt-broker/>

Geek For Geeks – Microservice Architecture – Introduction, Challenges & Best Practices. 06<sup>th</sup> November 2023. [Online] <https://www.geeksforgeeks.org/microservice-architecture-introduction-challenges-best-practices/>

Abyl – Navigating the 8 fallacies of distributed computing. Diaconu, Alex. [Online] <https://abyl.com/blog/8-fallacies-of-distributed-computing>

Medium – Mastering gRPC: The Ultimate Guide. Toksoz, Halil. 16<sup>th</sup> October 2023.  
[Online] <https://medium.com/@haliltoksz/mastering-grpc-the-ultimate-guide-9e6e65ff9f71>