Georgii Luppov

Table of contents 01
1. (gRPC) 02
3. (MQTT) 05
4. (REST) 10

1. (gRPC)

Sustainable Cities and Communities

Overall purpose:

The EmergencyService is designed to support urban safety as part of Sustainable Cities and Communities. It helps detect accidents quickly, identify them, allocate appropriate emergency teams, send all necessary information to the hospital before patients arrive to a hospital, and finds better routes for ambulances. This project aims to provide faster assistance to people and improve coordination between emergency services, creating safer, healthier, and more sustainable cities.

When accident happens, the request is sent to the server with all information from sensors/mobile apps etc. Request consist of location, timestamp, type of an accident, severity and additional description. For example, Traffic Accidents, Fires, Floods, Hurricanes, Medical Emergencies, Chemical spills, Radiation leaks (Unary). Server sends response that contains unique ID for every emergency and additional description.

Server analyses the case and sends multiple messages to specific emergency service or services, depending on the situation, that have already "subscribed" to this service (Clients). In order to subscribe they send serviceType and team ID, so server can assign a specific team to the emergency situation. For example, earthquake, server will notify ambulances and rescue teams. Server will send accidentID, location, accidentType, severity, and description (Server Stream).

When an ambulance is on the way to a hospital with a patient, it sends all information about that patient to the hospital, for instance, vital signs, and how much blood is required, so the hospital can prepare to accept the patient, because every second matters in case of emergency (Client Stream).

Meanwhile, the ambulance continuously sends its ID, location, timestamp, patient condition and to the server, which replies with dynamic route adjustments based on traffic conditions and hospital availability to minimize arrival time (Bidirectional Streaming RPC).

Operations:

```
// EmergencyService provides gRPC methods to manage emergencies
service EmergencyService {

  // 1. Unary gRPC for reporting a new accident with details
  rpc ReportAccident(AccidentRequest) returns (AccidentResponse);

  // 2. Server streaming gRPC to notify subscribed emergency services about relevant incidents
  rpc NotifyServices(ServiceSubscriptionRequest) returns (stream EmergencyNotification);

  // 3. Client streaming gRPC for ambulances to upload continuous patient data during transport
  rpc UploadPatientData(stream PatientData) returns (PatientReport);

  // 4. Bidirectional streaming gRPC for real-time route optimization and ambulance status updates
  rpc RouteOptimization(stream AmbulanceStatus) returns (stream RouteUpdate);
}
```

1. Unary RPC: ReportAccident

Allows clients (citizens, sensors, or apps) to report an accident by sending location, description, and timestamp data. Enables quick registration and classification of incidents with severity and accident type. The server generates a unique accident ID.

```
// 1. Unary
// Request message to report an accident
message AccidentRequest {
  string location = 1; // Location of the accident
  string description = 2; // Description of the accident
  int32 timestamp = 3; // Timestamp of the accident
  string accidentType = 4; // Type of accident (Traffic Accidents, Fires, Floods, Hurricanes, Medical Emergencies, Chemical spills, Radiation leaks)
  int32 severity = 5; // Severity level
}

// Response message after accident report
message AccidentResponse {
  string accidentId = 1; // Unique ID generated for the accident
  string message = 2; // Information about the accident
}
```

2. Server Streaming RPC: NotifyServices

Emergency services are “subscribed” to the server and it sends messages to appropriate service depending on the type of emergency. For example, if a fire is reported, the fire department and nearby ambulances will receive messages the incident location, severity, and response status. In case of a chemical spill, specialized teams will be notified in real time. This model ensures that each emergency service only receives the information which is up to their responsibilities, improving coordination and efficiency.

```
// 2. Server stream
// Request message for emergency services to subscribe to incident notifications
message ServiceSubscriptionRequest {
  string serviceType = 1; // Type of emergency service (FireDepartment, Ambulance, Police)
  string teamID = 2; // Number of the team that will be assigned
}

// Notification messages sent to subscribed emergency services
message EmergencyNotification {
  string accidentId = 1; // Associated accident ID
  string location = 2; // Incident location
  string accidentType = 3; // Type of accident
  int32 severity = 4; // Severity level
  string description = 5;  // Incident description
}
```

3. Client Streaming RPC: UploadPatientData

Enables an ambulance to send a stream of vital sign measurements of patient's data to the hospital during transportation. Allows hospitals to receive detailed patient information in real time and prepare before the patient's arrival.

```
// 3. Client stream
// Patient data streamed from ambulance to hospital
message PatientData {
  string patientId = 1; // Unique patient ID
  double heartRate = 2; // Heart rate in BPM
  double bloodPressure = 3; // Blood pressure data
  double oxygenSaturation = 4; // Oxygen info
  string additionalNotes = 5; // Any extra info from the ambulance
}

// Report after patient's data upload
message PatientReport {
  string report = 1; // Report message
}
```

4. Bidirectional Streaming RPC: RouteOptimization

Bidirectional stream where ambulances send live location/condition of the patient and the server sends updated route instructions to avoid traffic or changes hospital ER in case of heavy traffic or worsening patient's condition. Minimizes ambulance travel time by dynamically adapting routes based on traffic conditions, improving patient survival chances.

```
// 4. Bidirectional stream
// Information from the ambulance for route optimization
message AmbulanceStatus {
  string ambulanceId = 1; // Unique ambulance ID
  string location = 2; // Current location
  string patientCondition = 3; // Brief condition summary (stable, critical)
  int32 timestamp = 4; // Time of status update
}

// Route update messages that are sent back to the ambulance
message RouteUpdate {
  string newRouteInstructions = 1; // Text directions or encoded route data
  string estimatedArrivalTime = 2; // ETA to hospital
  string alternateHospital = 3; // Optional alternate hospital name
  int32 timestamp = 4; // Time route update sent
}
```

3. (MQTT)

The application monitors environmental factors such as weather warnings, air quality, and radon (radiation) levels within an urban environment to promote safer, healthier, and more resilient cities. Using MQTT messaging the system helps city authorities and citizens stay informed and take timely actions. MQTT makes it ideal for real time environmental monitoring. Its publish/subscribe model ensures that multiple subscribers (emergency services, mobile apps) receive updates instantly. Retained messages can ensure that even new subscribers immediately receive the latest environmental readings without delay. The MQTT topics use a 3 level hierarchy structured:
1. city/weather/warnings/Dublin
2. city/air/quality/Dublin-01
3. city/radiation/radon/Dublin-07

Publisher messages:

1. Weather Warning:
Topic: city/weather/warnings/Dublin
Message: "Red. Severe weather risk! Take precautions!"
The generateWeatherWarning() method is designed to simulate real time weather alerts for a city. It works by generating a random integer between 1 and 10, which is used to decide the severity of the weather warning. The logic categorizes this random number into three risk levels:
- Yellow (numbers 1–4): Indicates a low weather risk. The message advises people to stay alert, but there is no immediate danger.
- Orange (numbers 5–7): Represents a moderate weather risk. Conditions might be challenging, so extra care should be taken.
- Red (numbers 8–10): Signals a severe weather risk, so people should take precautions immediately to ensure safety.
Timely alerts can prevent accidents, reduce weather related harm, and increase city resilience.

```
private static String generateWeatherWarning() {
    int weatherRandom = (int) (Math.random() * 10) + 1;
    String warningLevel;
    if (weatherRandom < 5) {
        warningLevel = "Yellow. Low weather risk. Stay alert.";
    } else if (weatherRandom <= 7) {
        warningLevel = "Orange. Moderate weather risk.";
    } else {
        warningLevel = "Red. Severe weather risk! Take precautions!";
    }
    return warningLevel;
}
```

2. Air Quality Report:
Topic: city/air/quality/Dublin-01
Message: "Moderate air quality in Dublin 7"
The generateAirQuality() method simulates the current air quality index by selecting a random value between 1 and 5. Each value corresponds to air quality category: Good, Moderate, Unhealthy for Sensitive Groups, Unhealthy, Very Unhealthy. Depending on the random number, different

message is sent to the subsctibers. These categories allow city residents to take preventive actions, such as limiting outdoor activities during poor air quality conditions.

```
private static String generateAirQuality() {
    int airQualityIndexRandom = (int) (Math.random() * 5) + 1;
    String message;
    switch (airQualityIndexRandom){
        case 1:
            message = "Good air quality in Dublin 7";
            break;
        case 2:
            message = "Moderate air quality in Dublin 7";
            break;
        case 3:
            message = "Unhealthy for Sensitive Groups in Dublin 7";
            break;
        case 4:
            message = "Unhealthy air quality in Dublin 7";
            break;
        default:
            message = "Very Unhealthy air quality in Dublin 7";
    }
    return message;
}
```

3. Radon Level Status:
Topic: city/radiation/radon/Dublin-07
Message: "Radon level is safe in Dublin 1. Level: 3.2"
The generateRadonLevel() method simulates indoor radon gas measurements. It generates a random double value between 1.0 and 10.0 and classifies it as either safe (≤ 5.0) or unsafe (> 5.0). Radon is a health hazard especially in Ireland, and timely notifications can help residents take immediate actions, such as ventilation or professional testing.

```
private static String generateRadonLevel() {
    double radonLevelRandom = 1 + (Math.random() * 9.0);
    String status;
    if (radonLevelRandom <= 5){
        status = "Radon level is safe in Dublin 1. Level: " + radonLevelRandom;
    } else {
        status = "Radon level is unsafe in Dublin 1. Level: " + radonLevelRandom;
    }
    return status;
}
```

Subscriber messages:

Means that the user subscribes to all subtopics, for example, different neighborhoods. The topic in publisher is: city/air/quality/Dublin-01, and subscriber gets notifications about air quality from every area in the city: city/air/quality/# (Dublin-01, Dublin-02 etc).

1. Receiving weather warnings on city/weather/warnings/#:
"Yellow. Low weather risk. Stay alert."
This ensures residents receive early alerts to prepare for storms or extreme heat, reducing potential harm and disruption in the city.

```
sampleClient.subscribe("city/weather/warnings/#");
```

2. Receiving air quality updates on city/air/quality/#:
"Very Unhealthy air quality in Dublin 7
Citizens can take preventive actions such as staying indoors, using air filters, or wearing masks, reducing exposure to harmful pollutants.

```
sampleClient.subscribe("city/air/quality/#");
```

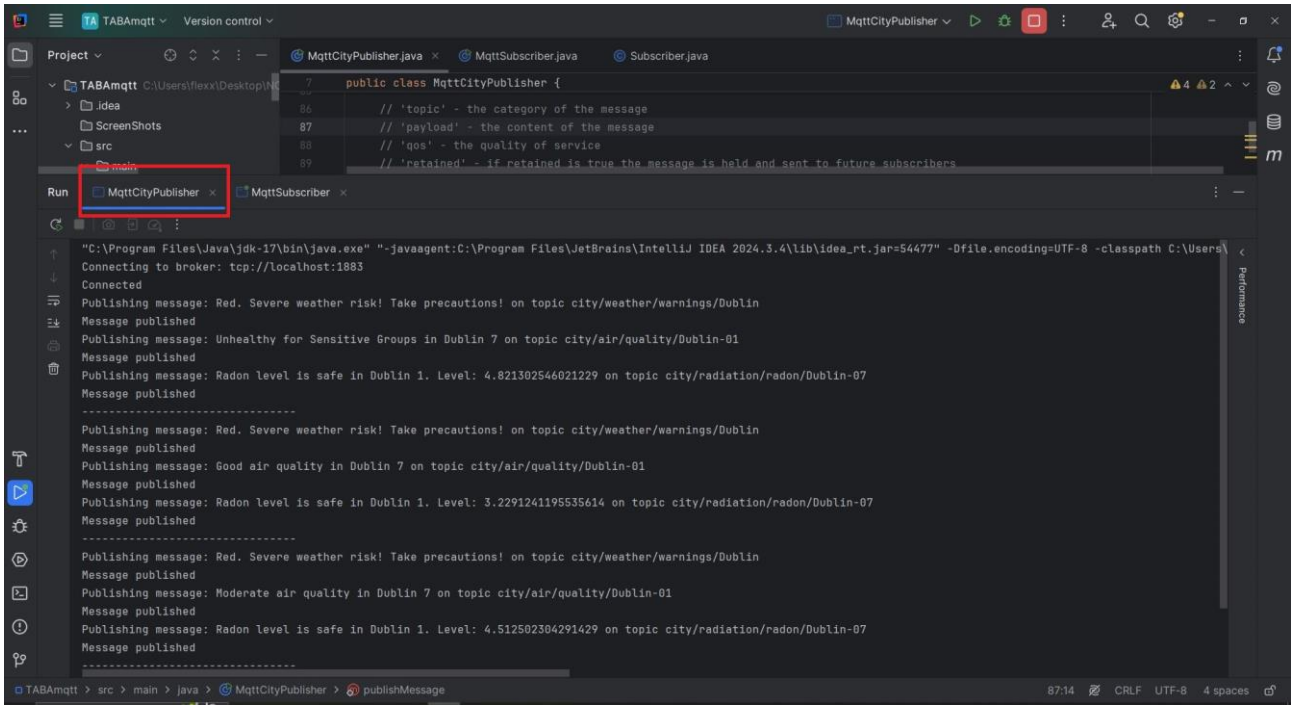
3. Receiving radon level alerts on city/radiation/radon/#:
"Radon level is safe in Dublin 1. Level: 4.2"
Residents can be warned about radon exposure risks in their districts/houses, enabling timely mitigation.

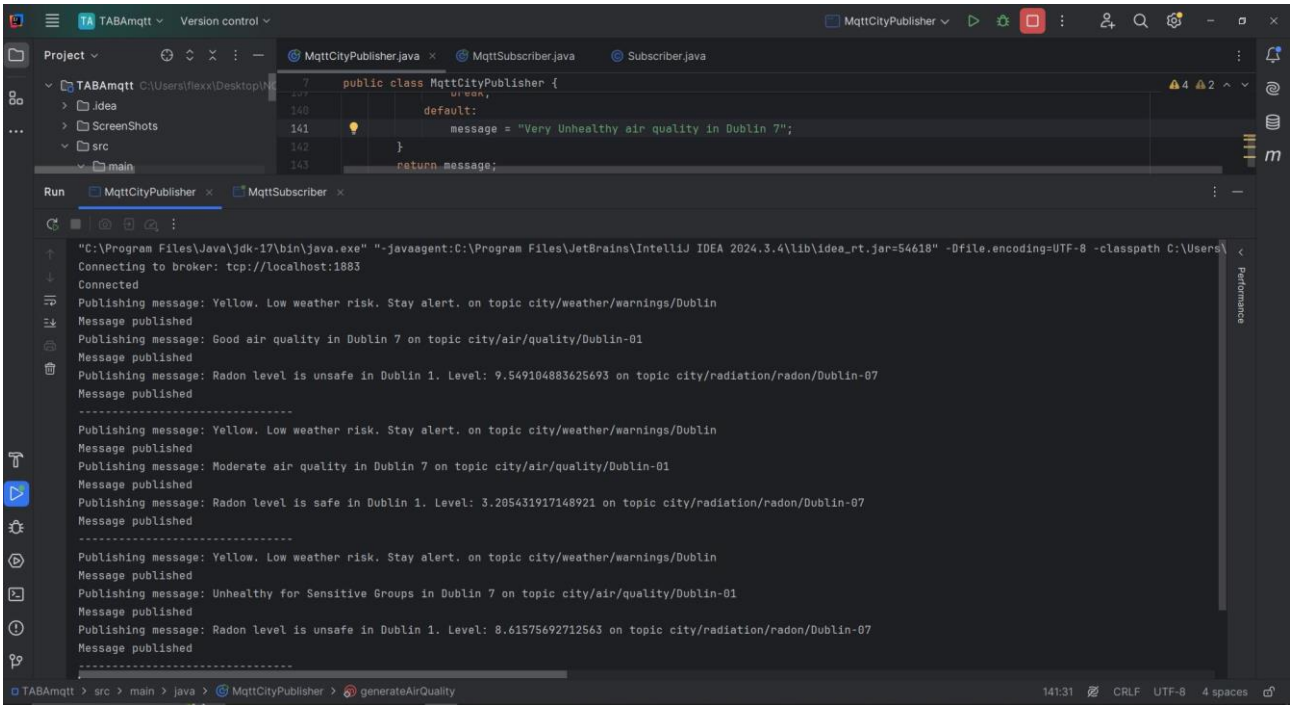
```
sampleClient.subscribe("city/radiation/radon/#");
```

```
// Called when a message arrives from the broker for a subscribed topic
@Override
public void messageArrived(String topic, MqttMessage message) throws Exception {
    System.out.println(new String(message.getPayload()) + " arrived from topic " + topic + " with qos " + message.getQos());
}
```

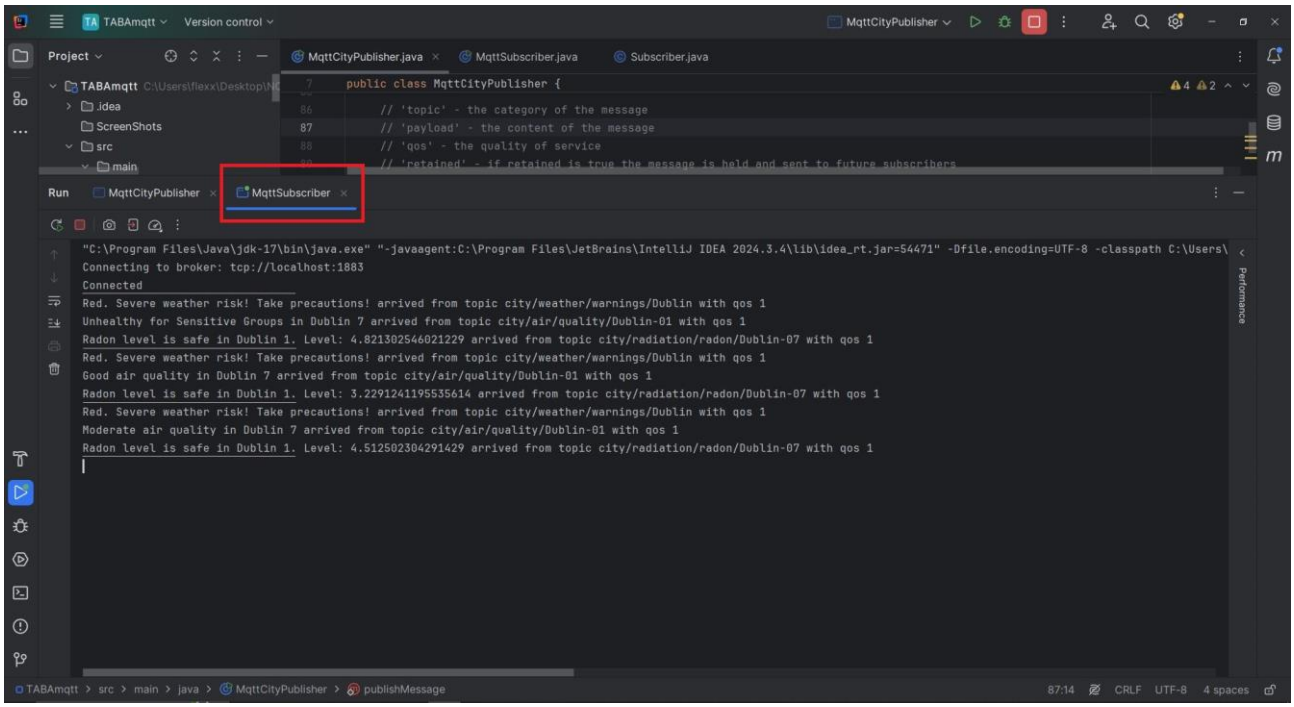
Examples of running the application:

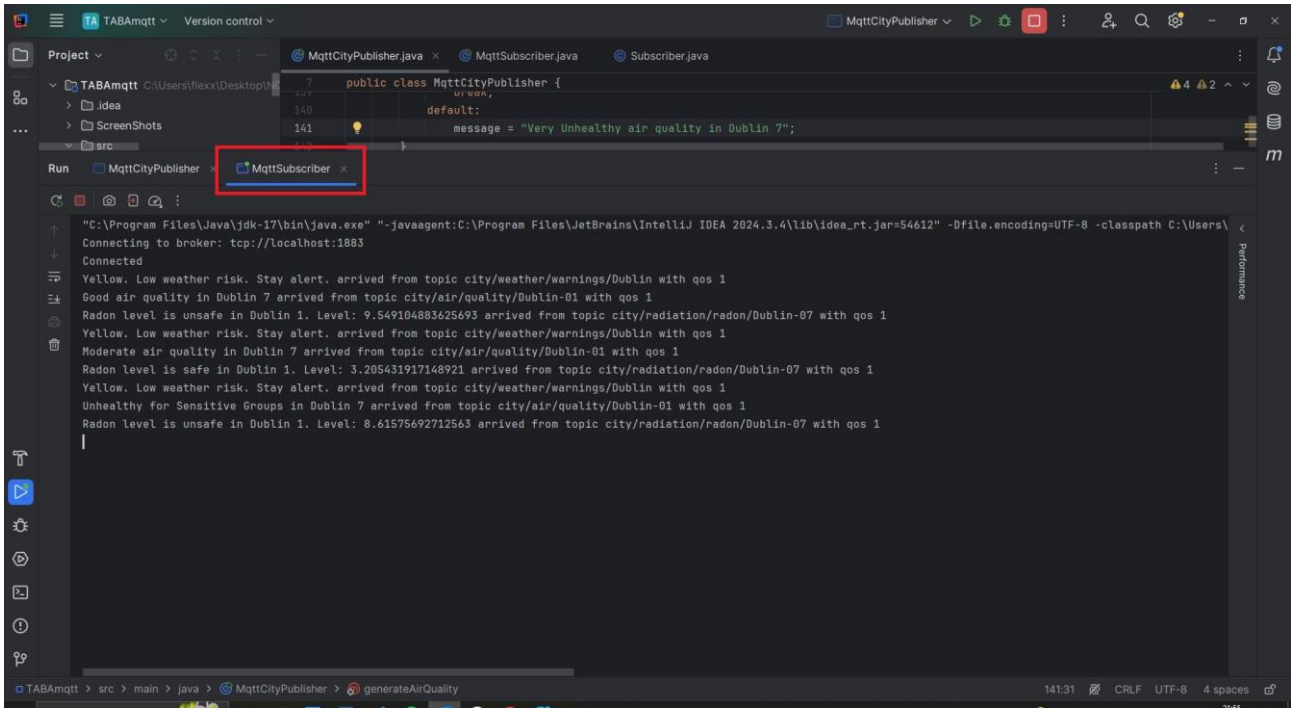
Publisher:





Subscriber:


TABAmqtt
Version control
MqttCityPublisher
Project
MqttCityPublisher.java
MqttSubscriber.java
Subscriber.java
public class MqttCityPublisher {
// 'topic' - the category of the message
// 'payload' - the content of the message
// 'qos' - the quality of service
// 'retained' - if retained is true the message is held and sent to future subscribers
Run
MqttCityPublisher
MqttSubscriber
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2024.3.4\lib\idea_rt.jar=54471" -Dfile.encoding=UTF-8 -classpath C:\Users\
Connecting to broker: tcp://localhost:1883
Connected
Red. Severe weather risk! Take precautions! arrived from topic city/weather/warnings/Dublin with qos 1
Unhealthy for Sensitive Groups in Dublin 7 arrived from topic city/air/quality/Dublin-01 with qos 1
Radon level is safe in Dublin 1. Level: 4.821302546021229 arrived from topic city/radiation/radon/Dublin-07 with qos 1
Red. Severe weather risk! Take precautions! arrived from topic city/weather/warnings/Dublin with qos 1
Good air quality in Dublin 7 arrived from topic city/air/quality/Dublin-01 with qos 1
Radon level is safe in Dublin 1. Level: 3.2291241195535614 arrived from topic city/radiation/radon/Dublin-07 with qos 1
Red. Severe weather risk! Take precautions! arrived from topic city/weather/warnings/Dublin with qos 1
Moderate air quality in Dublin 7 arrived from topic city/air/quality/Dublin-01 with qos 1
Radon level is safe in Dublin 1. Level: 4.512502304291429 arrived from topic city/radiation/radon/Dublin-07 with qos 1
Performance
TABAmqtt › src › main › java › MqttCityPublisher › publishMessage
87:14
CRLF
UTF-8
4 spaces


TABAmqtt
Version control
MqttCityPublisher
Project
MqttCityPublisher.java
MqttSubscriber.java
Subscriber.java
public class MqttCityPublisher {
default:
message = "Very Unhealthy air quality in Dublin 7";
Run
MqttCityPublisher
MqttSubscriber
"C:\Program Files\Java\jdk-17\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2024.3.4\lib\idea_rt.jar=54612" -Dfile.encoding=UTF-8 -classpath C:\Users\
Connecting to broker: tcp://localhost:1883
Connected
Yellow. Low weather risk. Stay alert. arrived from topic city/weather/warnings/Dublin with qos 1
Good air quality in Dublin 7 arrived from topic city/air/quality/Dublin-01 with qos 1
Radon level is unsafe in Dublin 1. Level: 9.549104883625693 arrived from topic city/radiation/radon/Dublin-07 with qos 1
Yellow. Low weather risk. Stay alert. arrived from topic city/weather/warnings/Dublin with qos 1
Moderate air quality in Dublin 7 arrived from topic city/air/quality/Dublin-01 with qos 1
Radon level is safe in Dublin 1. Level: 3.205431917148921 arrived from topic city/radiation/radon/Dublin-07 with qos 1
Yellow. Low weather risk. Stay alert. arrived from topic city/weather/warnings/Dublin with qos 1
Unhealthy for Sensitive Groups in Dublin 7 arrived from topic city/air/quality/Dublin-01 with qos 1
Radon level is unsafe in Dublin 1. Level: 8.61575692712563 arrived from topic city/radiation/radon/Dublin-07 with qos 1
Performance
TABAmqtt › src › main › java › MqttCityPublisher › generateAirQuality
141:31
CRLF
UTF-8
4 spaces

4. (REST)

This project is a RESTful Web Service to support a public transport organization in improving urban mobility. The service provides information about bus routes and real time locations of buses, helping passengers plan their journeys more efficiently. Additionally, it allows users to submit feedback about the service, enabling the organization to address issues and improve quality. This system contributes to sustainable development goals related to sustainable cities and communities.

Endpoints and implementation:

1. GET /api/routes

Returns a list of available bus routes, helps users discover available routes to plan their trips.

```
// GET /api//routes
// Returns a list of predefined routes
@GetMapping("/routes")
public List<Route> getRoutes() {
    ArrayList<Route> routes = new ArrayList<>();
    // Creating and adding sample Route objects to the list
    Route route1 = new Route(1, "Route 101", "Bus");
    routes.add(route1);
    Route route2 = new Route(2, "Route 102", "Bus");
    routes.add(route2);
    Route route3 = new Route(3, "Route 103", "Bus");
    routes.add(route3);
    return routes;
}
```

2. GET /api/routes/busesLive

Provides live data on buses currently operating on randomly selected routes, and gives real time location updates to passengers, improving travel planning and experience.

```
// GET /api//routes/busesLive
// Returns a list of buses currently running on a randomly selected route
@GetMapping("/routes/busesLive")
public List<BusLocation> getBusesOnRoute() {
    ArrayList<BusLocation> buses = new ArrayList<>();
    // Select a random route number between 1 and 3
    int randomRoute = (int) (Math.random() * 3) + 1;
    // Buses location info depending on the random route
    if (randomRoute == 1) {
        buses.add(new BusLocation(101, 53.347625305418326, -6.2590545814028005, "On Time"));
    } else if (randomRoute == 2) {
        buses.add(new BusLocation(201, 53.34885852669177, -6.237723634360727, "On Time"));
    } else {
        buses.add(new BusLocation(301, 53.34804160690227, -6.247478167395365, "Delayed"));
    }
    return buses;
}
```

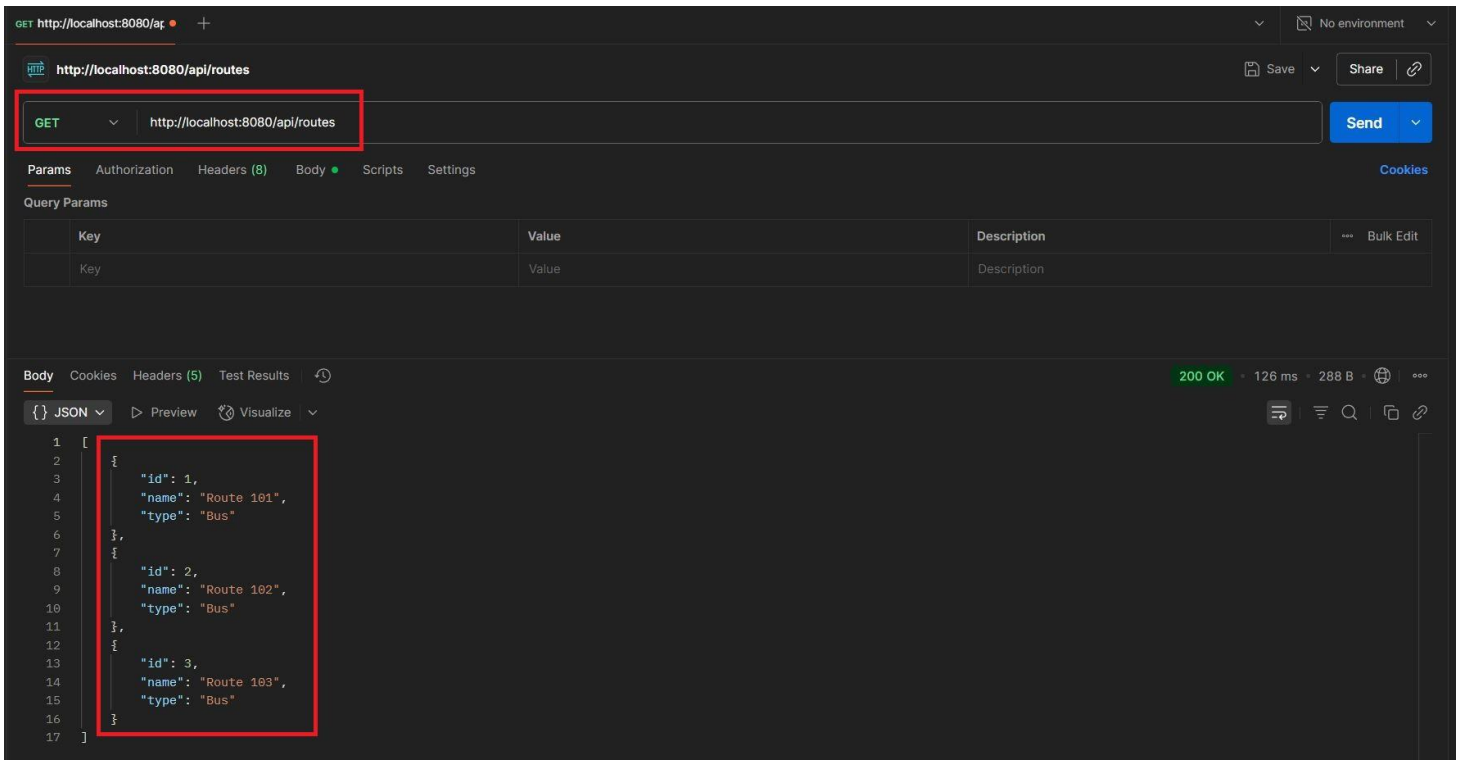
3. POST /api/feedback
Accepts user feedback about buses or routes in JSON format, collects passenger feedback to improve service quality.

```java
// POST /api/feedback
// Accepts feedback data in the request body and returns a confirmation message
@PostMapping("/feedback")
public String postFeedback(@RequestBody Feedback feedback) {
    System.out.println(LocalTime.now() + "Feedback submitted successfully.");
    return "Feedback submitted successfully. " + feedback;
}
```

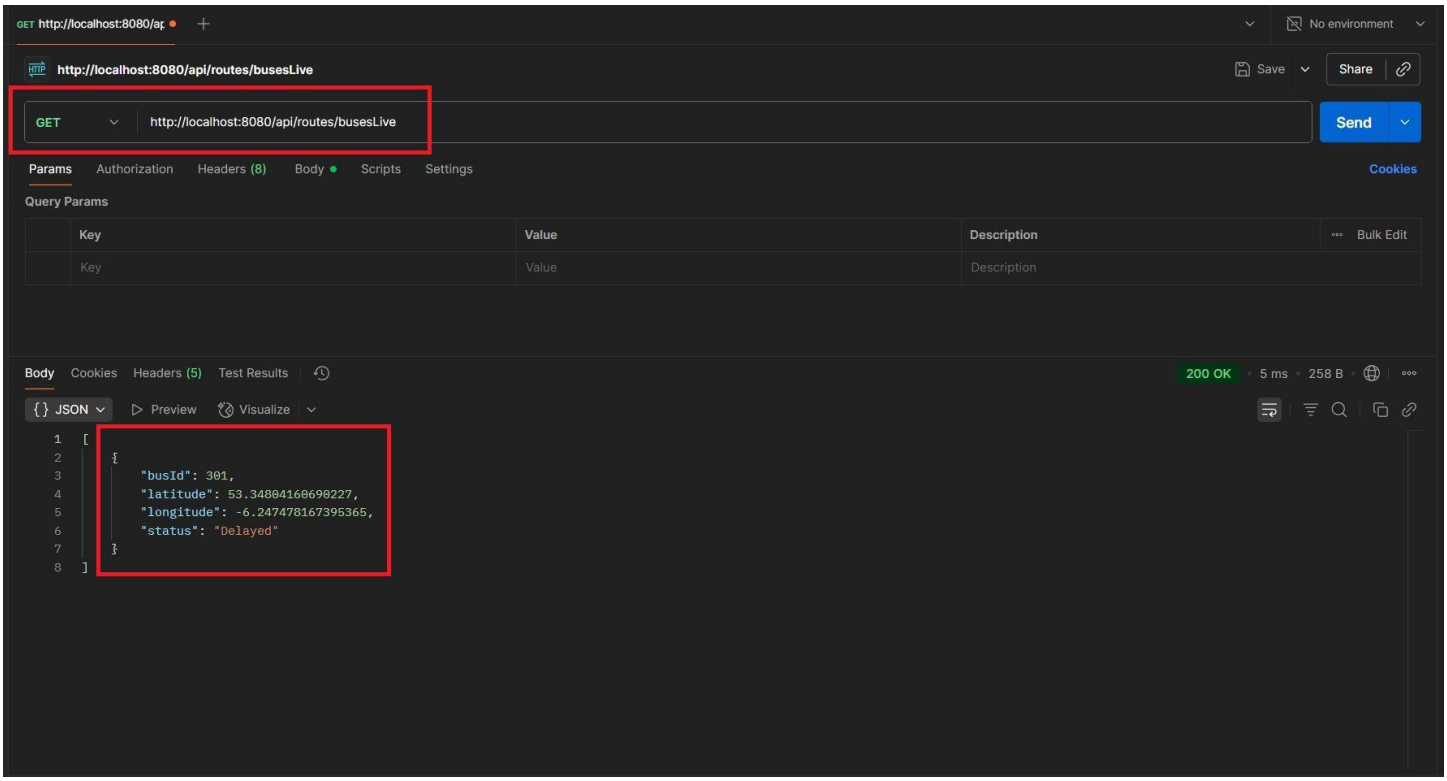
```java
// Override toString to print info later
@Override
public String toString() {
    return "Feedback{" +
            "user='" + user + '\'' +
            ", routeId=" + routeId +
            ", busId=" + busId +
            ", message='" + message + '\'' +
            '}';
}
```

POSTMAN:

GET /api/routes



GET /api/routes/busesLive



POST /api/feedback