Aplikacija se sastoji iz tri mikroservisa

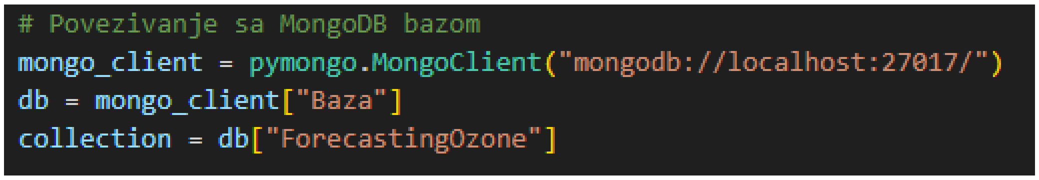
- **Sensor mikroservis**
- **Analytics mikroservis**
- **EventInfo mikroservis**

Sto se tice tehnologija koje su koriscenje, Sensor i Analytics mikroservisi su radjeni u Python-u, a EventInfo u .Net-u.

Sensor mikroservis

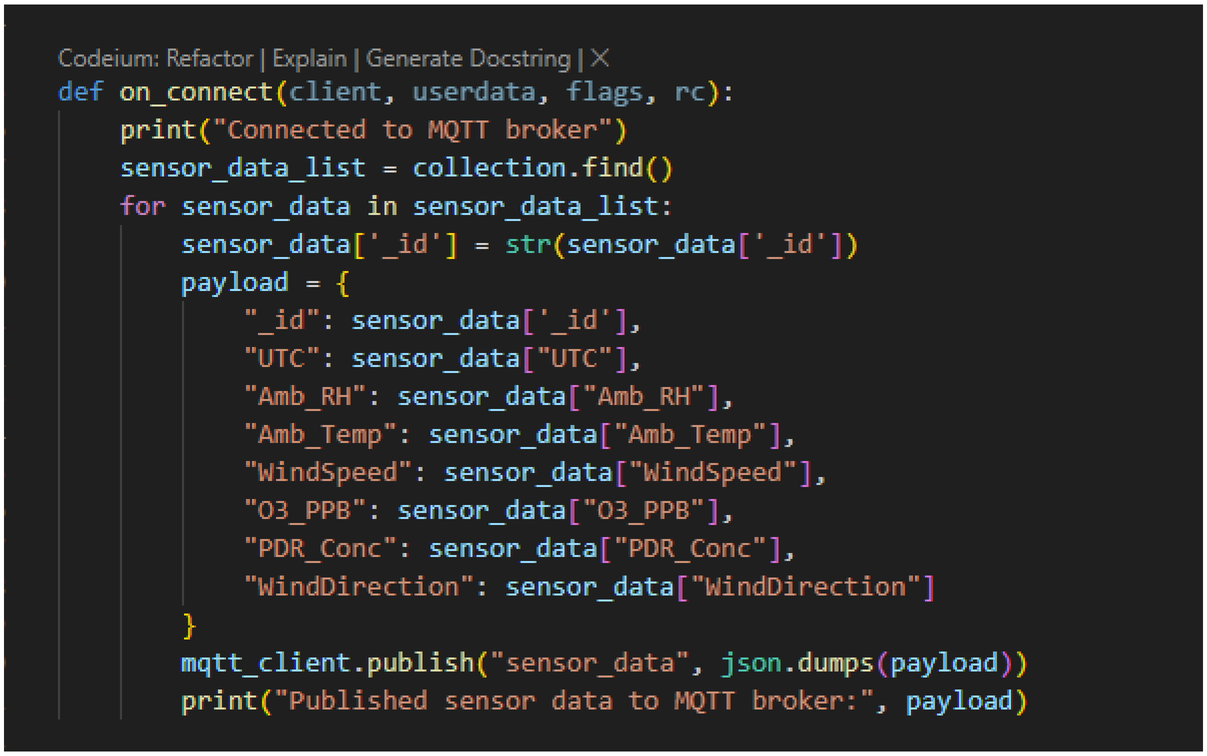
Funkcionalnosti:

- **Povezivanje sa MongoDB bazom**: Aplikacija se povezuje na lokalnu MongoDB bazu podataka kako bi preuzela podatke.



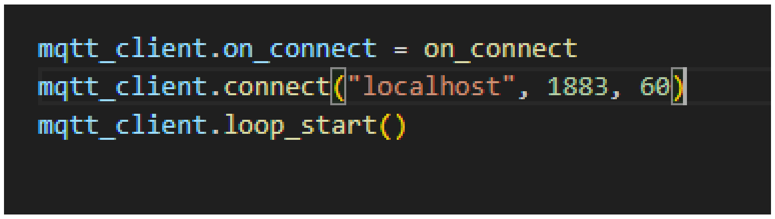
```
# Povezivanje sa MongoDB bazom
mongo_client = pymongo.MongoClient("mongodb://localhost:27017/")
db = mongo_client["Baza"]
collection = db["ForecastingOzone"]
```

- **Prenos podataka**: Podaci sa senzora, uključujući informacije o vlažnosti, temperaturi, brzini vetra, koncentraciji ozona i pravcu vetra, šalju se na MQTT temu u JSON formatu.



```
def on_connect(client, userdata, flags, rc):
    print("Connected to MQTT broker")
    sensor_data_list = collection.find()
    for sensor_data in sensor_data_list:
        sensor_data['_id'] = str(sensor_data['_id'])
        payload = {
            "_id": sensor_data['_id'],
            "UTC": sensor_data["UTC"],
            "Amb_RH": sensor_data["Amb_RH"],
            "Amb_Temp": sensor_data["Amb_Temp"],
            "WindSpeed": sensor_data["WindSpeed"],
            "O3_PPB": sensor_data["O3_PPB"],
            "PDR_Conc": sensor_data["PDR_Conc"],
            "WindDirection": sensor_data["WindDirection"]
        }
        mqtt_client.publish("sensor_data", json.dumps(payload))
        print("Published sensor data to MQTT broker:", payload)
```

- **Povezivanje sa MQTT brokerom**: Korišćenjem Paho MQTT biblioteke, aplikacija se povezuje na lokalni MQTT broker.

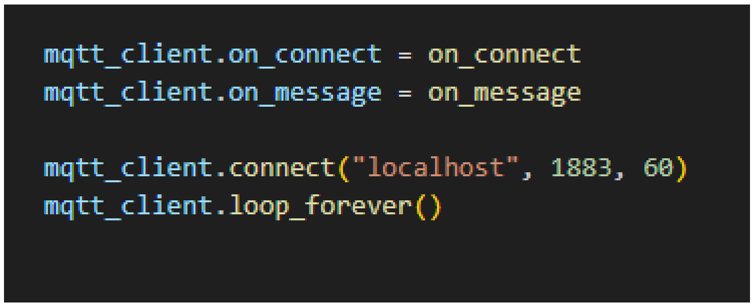


```
mqtt_client.on_connect = on_connect
mqtt_client.connect("localhost", 1883, 60)
mqtt_client.loop_start()
```

Analytics mikroservis

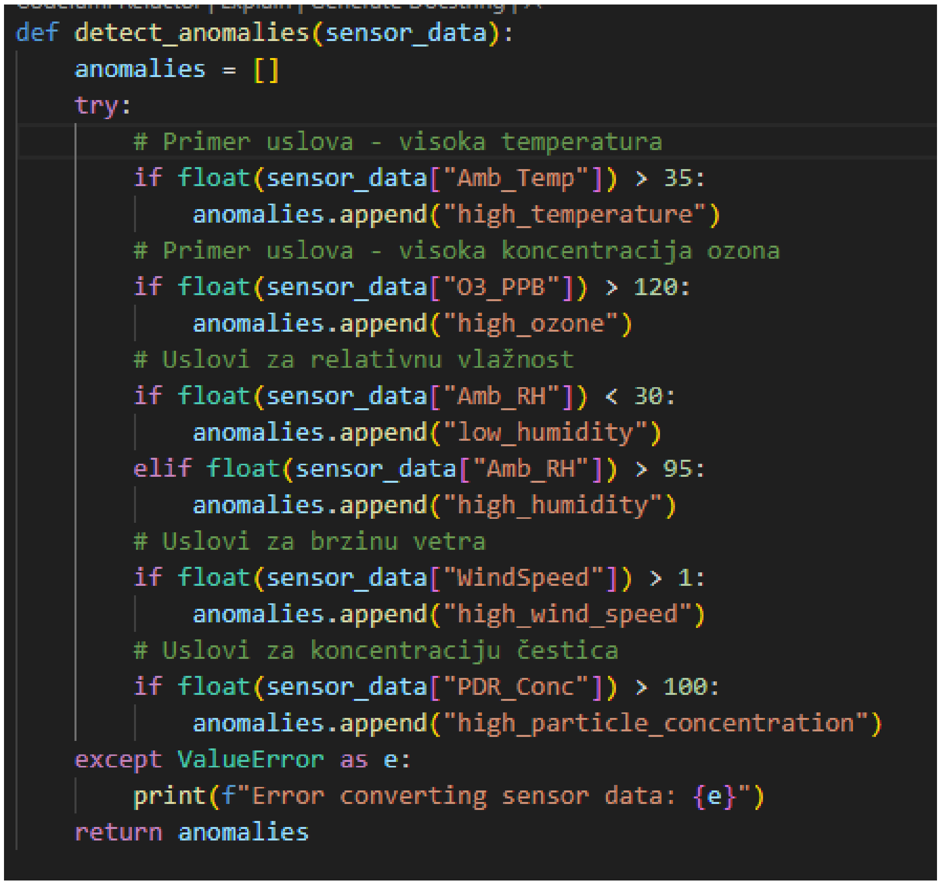
Funkcionalnosti:

- **Povezivanje sa MQTT brokerom**: Aplikacija se povezuje sa lokalnim MQTT brokerom za primanje i slanje podataka.



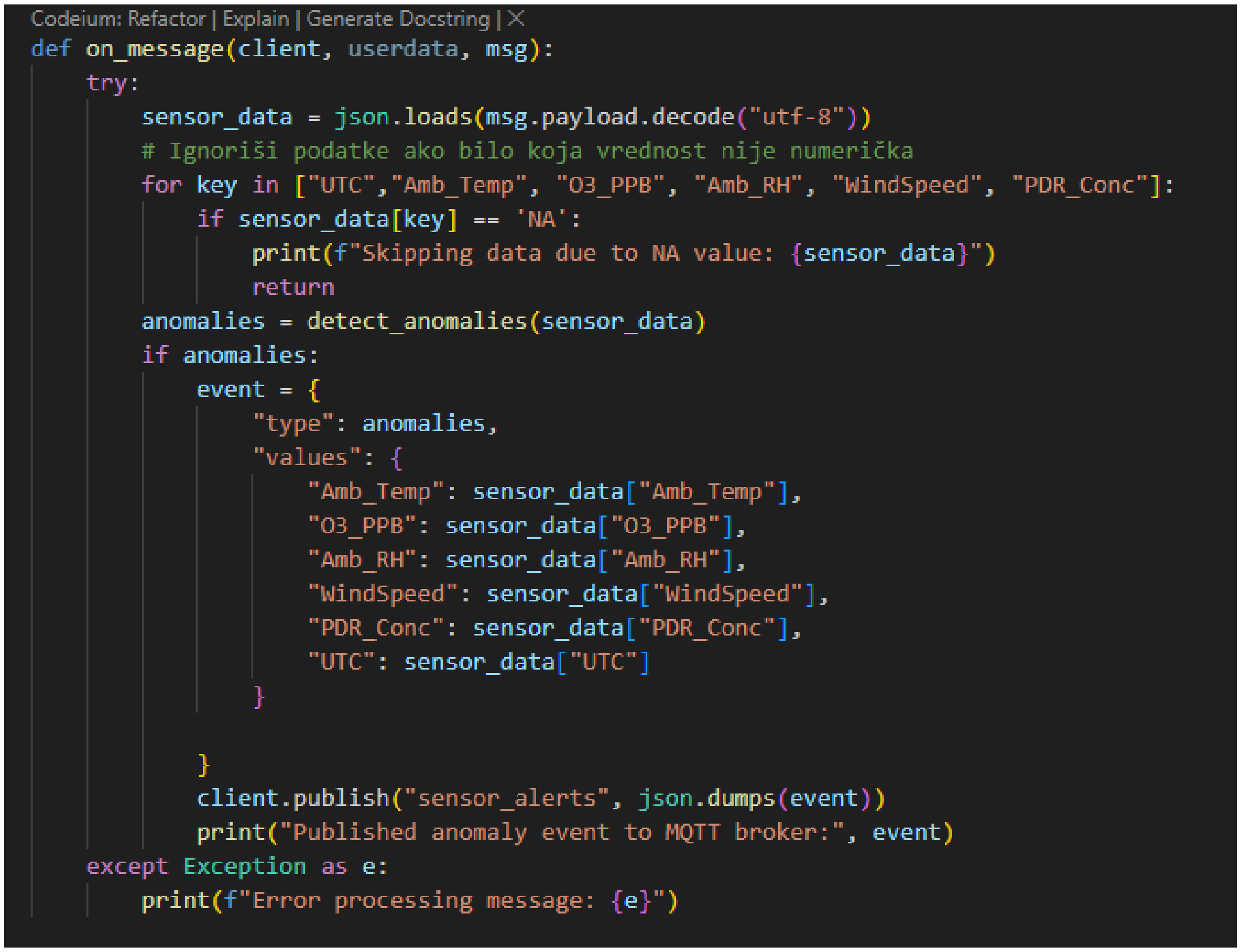
```
mqtt_client.on_connect = on_connect
mqtt_client.on_message = on_message

mqtt_client.connect("localhost", 1883, 60)
mqtt_client.loop_forever()
```

- **Detekcija anomalija**: Aplikacija analizira primljene podatke sa senzora i detektuje anomalije bazirane na predefinisanim uslovima.



```
def detect_anomalies(sensor_data):
    anomalies = []
    try:
        # Primer uslova - visoka temperatura
        if float(sensor_data["Amb_Temp"]) > 35:
            anomalies.append("high_temperature")
        # Primer uslova - visoka koncentracija ozona
        if float(sensor_data["O3_PPB"]) > 120:
            anomalies.append("high_ozone")
        # Uslovi za relativnu vlažnost
        if float(sensor_data["Amb_RH"]) < 30:
            anomalies.append("low_humidity")
        elif float(sensor_data["Amb_RH"]) > 95:
            anomalies.append("high_humidity")
        # Uslovi za brzinu vetra
        if float(sensor_data["WindSpeed"]) > 1:
            anomalies.append("high_wind_speed")
        # Uslovi za koncentraciju čestica
        if float(sensor_data["PDR_Conc"]) > 100:
            anomalies.append("high_particle_concentration")
    except ValueError as e:
        print(f"Error converting sensor data: {e}")
    return anomalies
```

- **Slanje upozorenja**: Kada se detektuje anomalija, aplikacija šalje upozorenje na određenu MQTT temu.

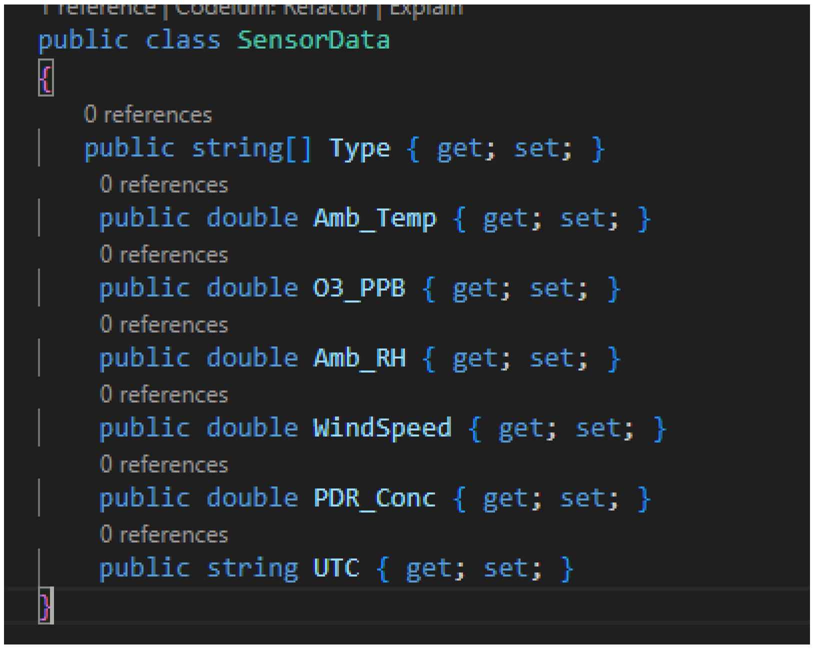


```
Codeium: Refactor | Explain | Generate Docstring | X
def on_message(client, userdata, msg):
    try:
        sensor_data = json.loads(msg.payload.decode("utf-8"))
        # Ignoriši podatke ako bilo koja vrednost nije numerička
        for key in ["UTC","Amb_Temp", "O3_PPB", "Amb_RH", "WindSpeed", "PDR_Conc"]:
            if sensor_data[key] == 'NA':
                print(f"Skipping data due to NA value: {sensor_data}")
                return
        anomalies = detect_anomalies(sensor_data)
        if anomalies:
            event = {
                "type": anomalies,
                "values": {
                    "Amb_Temp": sensor_data["Amb_Temp"],
                    "O3_PPB": sensor_data["O3_PPB"],
                    "Amb_RH": sensor_data["Amb_RH"],
                    "WindSpeed": sensor_data["WindSpeed"],
                    "PDR_Conc": sensor_data["PDR_Conc"],
                    "UTC": sensor_data["UTC"]
                }

            }
            client.publish("sensor_alerts", json.dumps(event))
            print("Published anomaly event to MQTT broker:", event)
    except Exception as e:
        print(f"Error processing message: {e}")
```

EventInfo mikroservis

Funkcionalnosti kontrolera:

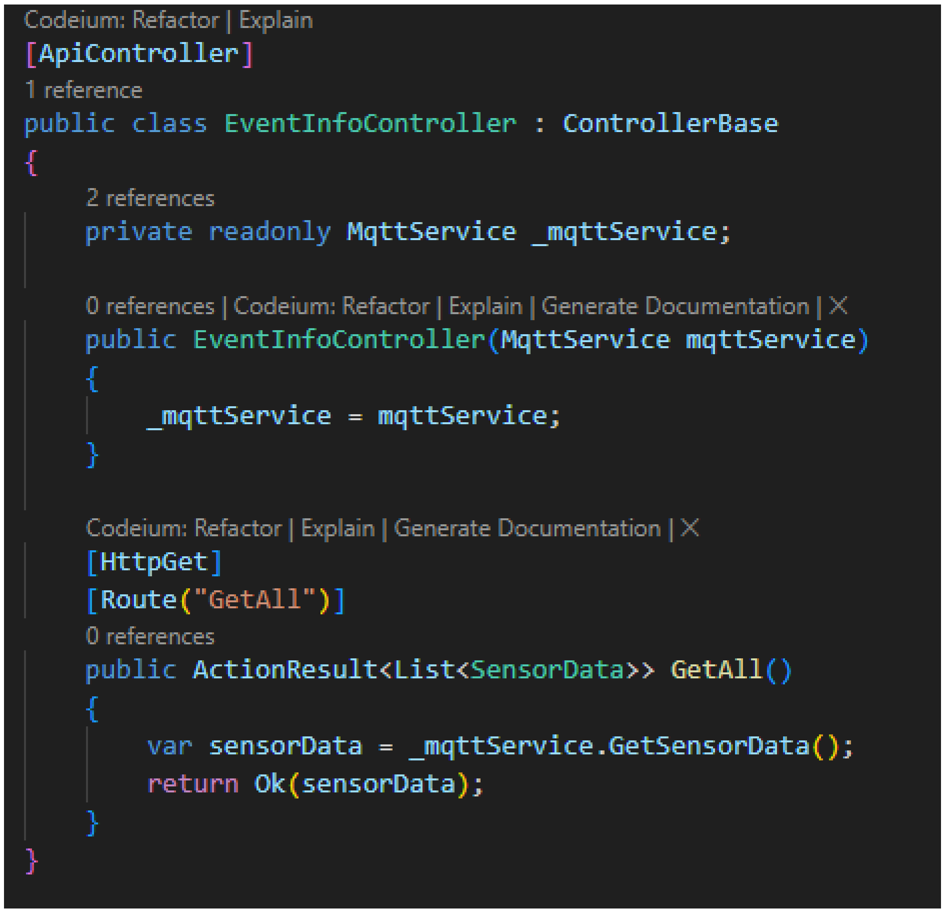
- **Definisanje klase SensorData**: Klasa predstavlja podatke senzora koje će API izlagati.



```
1 reference | Codeium: Refactor | Explain
public class SensorData
{
    0 references
    public string[] Type { get; set; }
     0 references
     public double Amb_Temp { get; set; }
     0 references
     public double O3_PPB { get; set; }
     0 references
     public double Amb_RH { get; set; }
     0 references
     public double WindSpeed { get; set; }
     0 references
     public double PDR_Conc { get; set; }
     0 references
     public string UTC { get; set; }
}
```

- **Kontroler EventInfoController**:

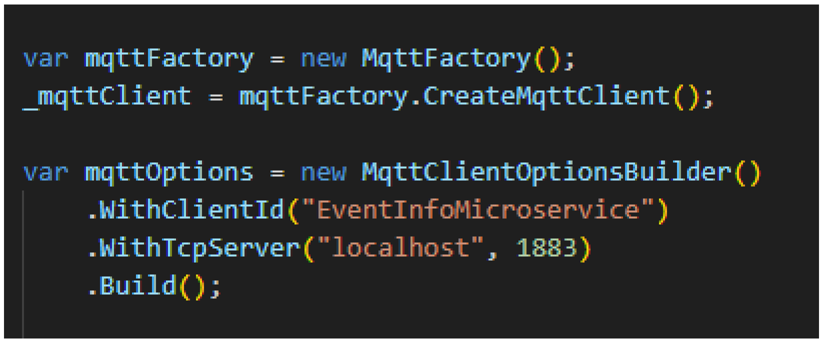
    - [HttpGet] [Route("GetAll")]: Definiše GET medtodu GetAll koji vraća listu svih prikupljenih podataka senzora.



```
Codeium: Refactor | Explain
[ApiController]
1 reference
public class EventInfoController : ControllerBase
{
    2 references
    private readonly MqttService _mqttService;

    0 references | Codeium: Refactor | Explain | Generate Documentation | ×
    public EventInfoController(MqttService mqttService)
    {
        _mqttService = mqttService;
    }

    Codeium: Refactor | Explain | Generate Documentation | ×
    [HttpGet]
    [Route("GetAll")]
    0 references
    public ActionResult<List<SensorData>> GetAll()
    {
        var sensorData = _mqttService.GetSensorData();
        return Ok(sensorData);
    }
}
```

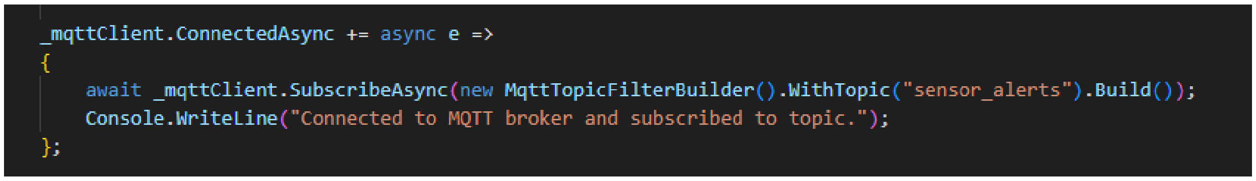
Funkcionalnosti servisa:

- Kreiranje Mqtt klijenta i postavljanje opcija za povezivanje



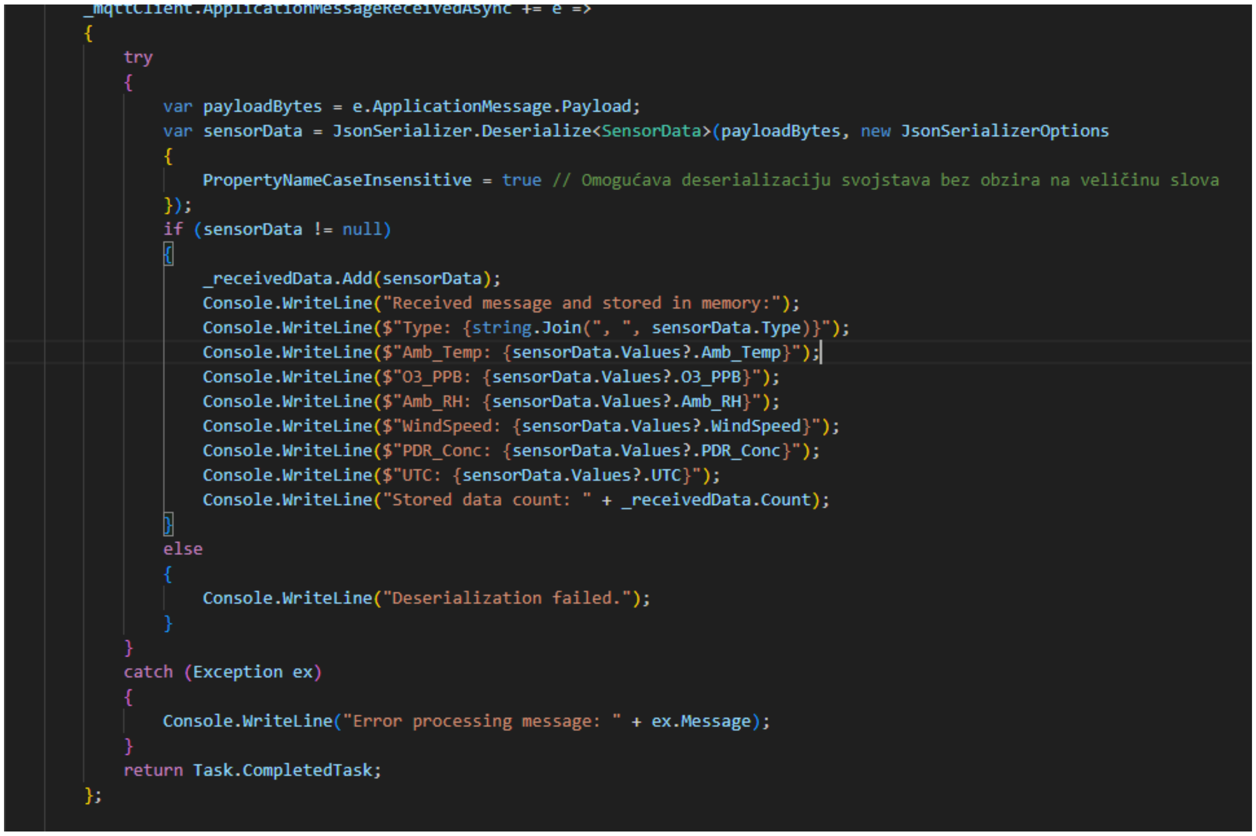
```
var mqttFactory = new MqttFactory();
_mqttClient = mqttFactory.CreateMqttClient();

var mqttOptions = new MqttClientOptionsBuilder()
    .WithClientId("EventInfoMicroservice")
    .WithTcpServer("localhost", 1883)
    .Build();
```

- **Definisanje handlera za događaj povezivanja**: Kada se klijent poveže, pretplaćuje se na temu sensor_alerts.



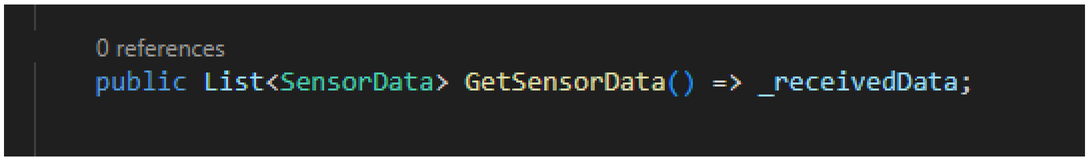
```
_mqttClient.ConnectedAsync += async e =>
{
    await _mqttClient.SubscribeAsync(new MqttTopicFilterBuilder().WithTopic("sensor_alerts").Build());
    Console.WriteLine("Connected to MQTT broker and subscribed to topic.");
};
```

- **Definisanje handlera za prijem poruka**: Kada klijent primi poruku, deserializuje podatke i dodaje ih u _receivedData listu.



```
_mqttClient.ApplicationMessageReceivedAsync += e =>
{
    try
    {
        var payloadBytes = e.ApplicationMessage.Payload;
        var sensorData = JsonSerializer.Deserialize<SensorData>(payloadBytes, new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true // Omogućava deserializaciju svojstava bez obzira na veličinu slova
        });
        if (sensorData != null)
        {
            _receivedData.Add(sensorData);
            Console.WriteLine("Received message and stored in memory:");
            Console.WriteLine($"Type: {string.Join(", ", sensorData.Type)}");
            Console.WriteLine($"Amb_Temp: {sensorData.Values?.Amb_Temp}");
            Console.WriteLine($"O3_PPB: {sensorData.Values?.O3_PPB}");
            Console.WriteLine($"Amb_RH: {sensorData.Values?.Amb_RH}");
            Console.WriteLine($"WindSpeed: {sensorData.Values?.WindSpeed}");
            Console.WriteLine($"PDR_Conc: {sensorData.Values?.PDR_Conc}");
            Console.WriteLine($"UTC: {sensorData.Values?.UTC}");
            Console.WriteLine("Stored data count: " + _receivedData.Count);
        }
        else
        {
            Console.WriteLine("Deserialization failed.");
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error processing message: " + ex.Message);
    }
    return Task.CompletedTask;
};
```

- **Smesta podatke u listu SensorData**



```
0 references
public List<SensorData> GetSensorData() => _receivedData;
```

- Klasa SensorData predstavlja podatke prikupljene sa senzora. Ova klasa sadrži dva svojstva:

    1. **Type**: Niz stringova (string[]) koji opisuje tipove anomalija.
    2. **Values**: Instanca klase Values koja sadrži konkretne merne vrednosti sa senzora.