Guse Ovidiu-Marius

Pre-Practica BEIA

Guse Ovidiu-Marius

Hello! This is my Google Doc for my Pre-Practice/Internship Project:

Tasklist:

- IoT device connection via MQTT to the database.
- Use Node-Red for data processing.
- Local storage in a personal file using Node-Red.
- Data visualization in Grafana.
- Setting notifications and alerts in Grafana, receiving messages on Telegram.
- Data processing using Node-Red.
- Create and test smart contracts on Blockchain.
- Data transmission via Telegram chatbot.
- Query data using chatbot.
-

Firstly, I started with creating a program in PyCharm that receives the temperature and air pollution (used the latitude and longitude for Bucharest) from the website https://openweathermap.org using an API.

```
import pip._vendor.requests as requests

import json

import time

import paho.mqtt.publish as publish

# Setările brokerului MQTT

broker = "mqtt.beia-telemetrie.ro"

port = 1883

topic = "training/device/Guse-Ovidiu"

topic_processed="training/device/Guse-Ovidiu/temperature"
```

Guse Ovidiu-Marius

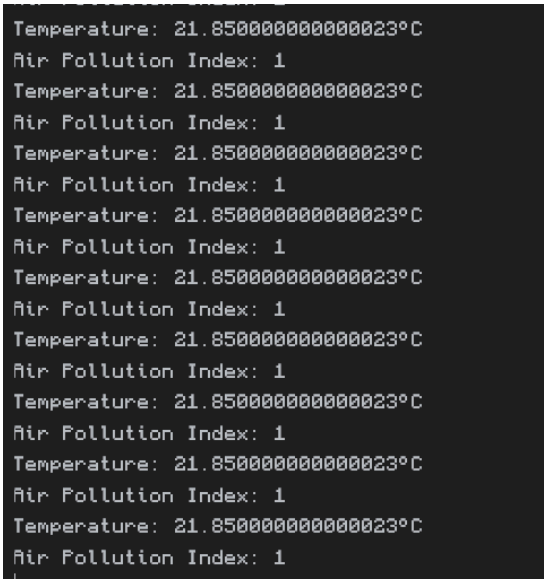
```
# Coordonatele geografice

latitude = 44.426

longitude = 26.102

# Cheia API OpenWeatherMap

api_key = "961dbbf41caa9f205be694bbb208a8f5"

# URL pentru obținerea datelor meteorologice

weather_url =
f"https://api.openweathermap.org/data/2.5/weather?lat={latitude}&lon={longitude}&appid={ap
i_key}"

# URL pentru obținerea datelor de poluare a aerului

pollution_url =
f"https://api.openweathermap.org/data/2.5/air_pollution?lat={latitude}&lon={longitude}&appid
={api_key}"

# Funția pentru obținerea temperaturii curente din API

def get_current_temperature():

    response = requests.get(weather_url)

    data = response.json()

    temperature_kelvin = data["main"]["temp"]

    temperature_celsius = temperature_kelvin - 273.15

    return temperature_celsius

# Funția pentru obținerea nivelului de poluare a aerului

def get_air_pollution():

    response = requests.get(pollution_url)

    data = response.json()

    air_pollution_index = data["list"][0]["main"]["aqi"]

    return air_pollution_index

# Publicarea datelor în mod continuu
```

Guse Ovidiu-Marius

```
while True:
    # Obținerea temperaturii curente
    temperature = get_current_temperature()
    print(f"Temperature: {temperature}°C")

    # Obținerea nivelului de poluare a aerului
    air_pollution = get_air_pollution()
    print(f"Air Pollution Index: {air_pollution}")

    payload_dict = {
        "temperature": temperature,
        "air_pollution": air_pollution
    }

    # Publicarea datelor în topic
    publish.single(topic, hostname=broker, port=port, payload=json.dumps(payload_dict))
    time.sleep(10)
```

Guse Ovidiu-Marius

On the terminal we can see the temperatures and air pollution for Bucharest:



Grafana:

Grafana is an open-source analytics and monitoring platform that allows you to create, visualize, and understand data through customizable dashboards.Grafana supports integration with various data sources, including databases (e.g., MySQL, PostgreSQL), time-series databases (e.g., Prometheus, InfluxDB), cloud monitoring services (e.g., AWS CloudWatch, Google Cloud Monitoring), and more.

We have to create a new dashboard and to connect to a database, where the collected data will be stocated.

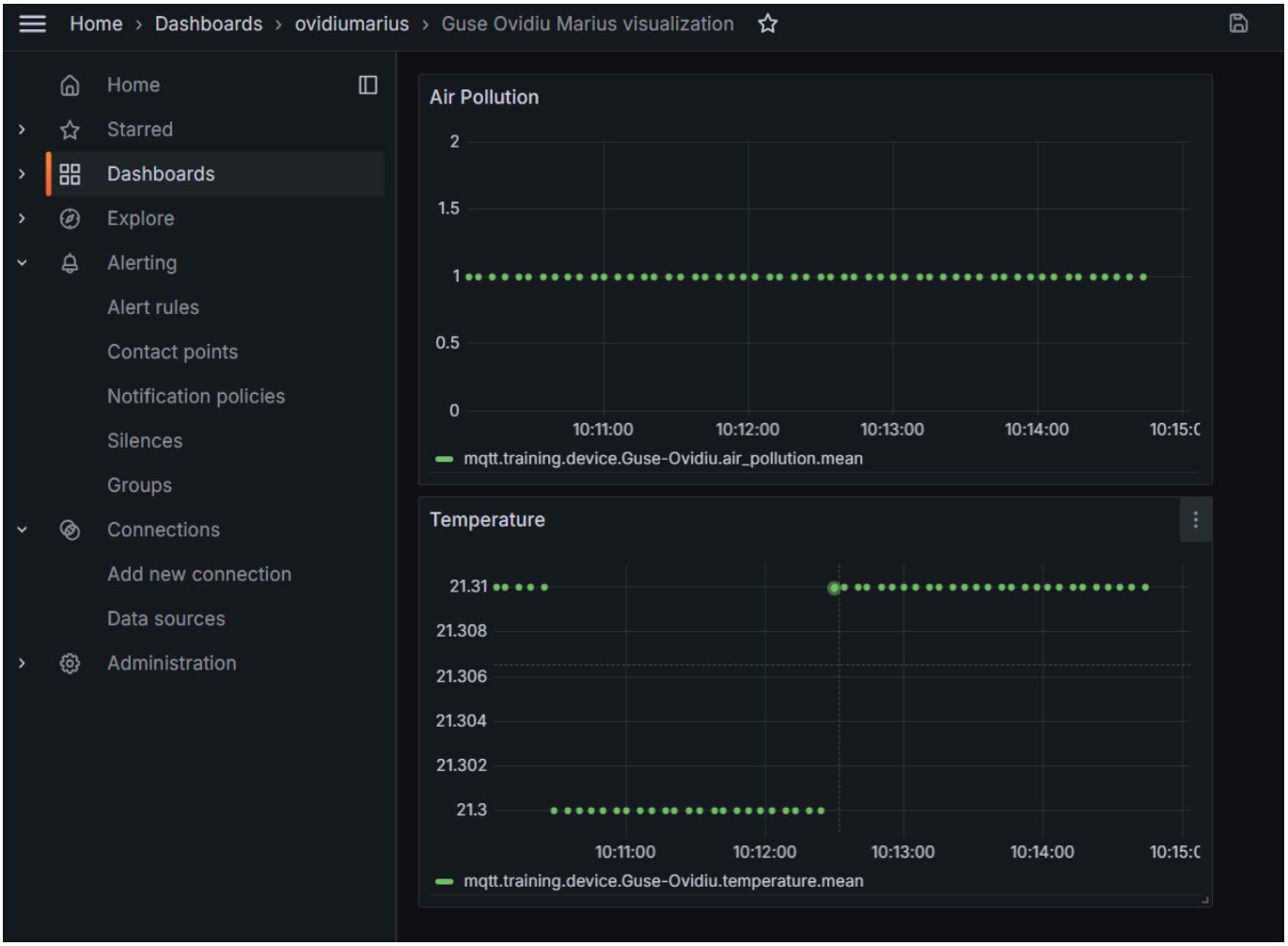
As we can see, I configured the measurement, by searching the topic of the MQTT broker. In this case, the topic is “training.device.Guse-Ovidiu” .

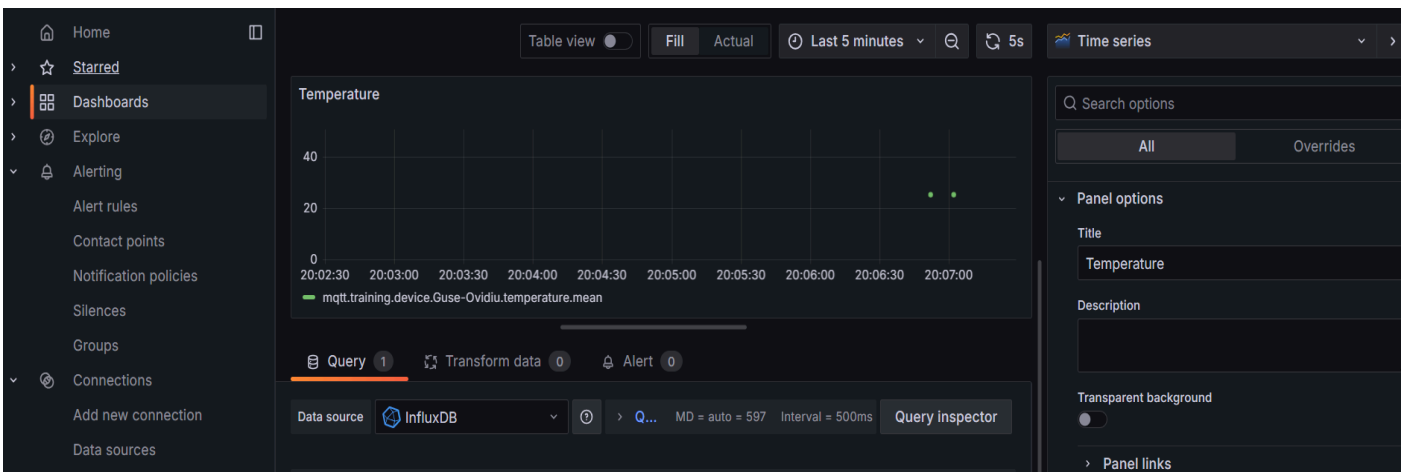
After this setting, the collected data is transmitted to a database, in our case, InfluxDB and the data is displayed on a graph.

This way I connected to mqtt.beia-telemetrie.ro on Grafana website so I could visualize in real time the temperature and the air pollution, these being added to my dashboard.

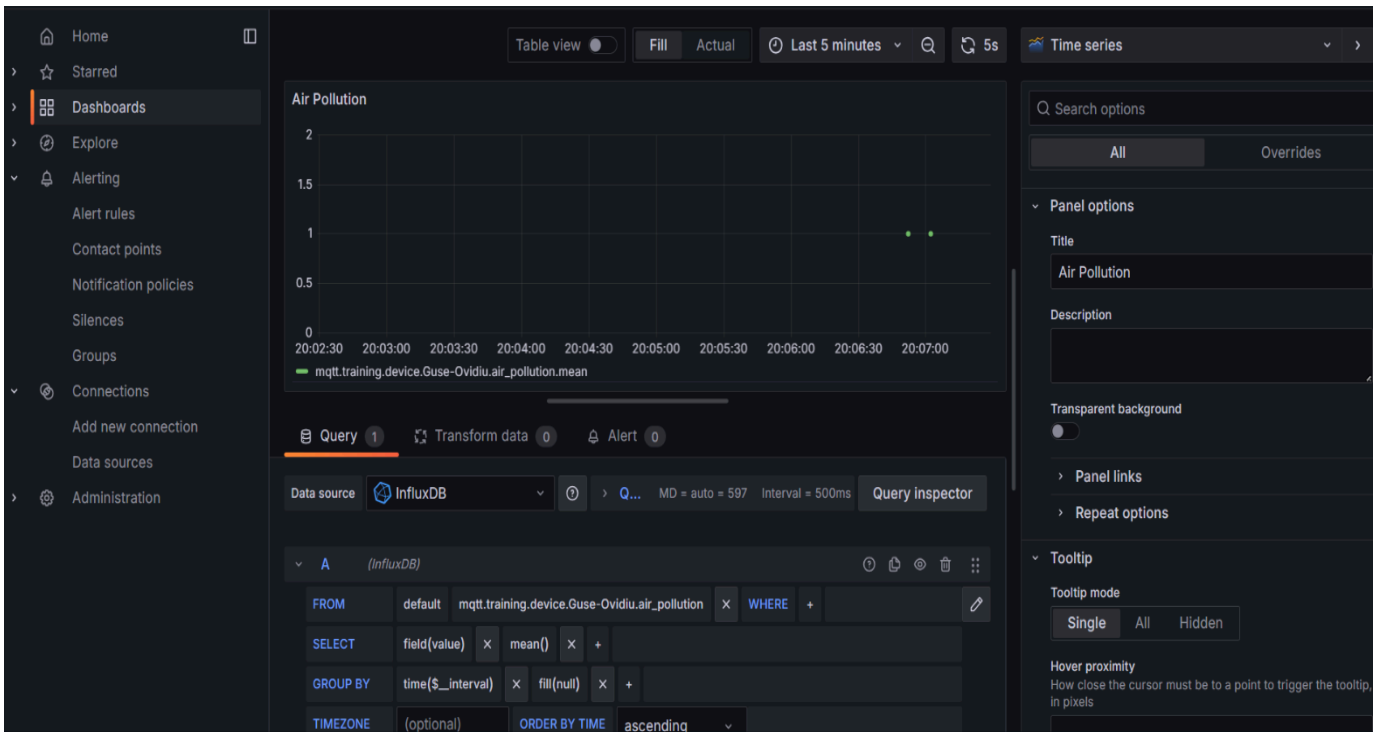
The datasource is **INFLUXDB.**

Guse Ovidiu-Marius





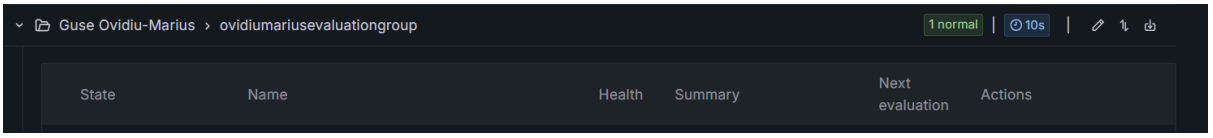
Guse Ovidiu-Marius



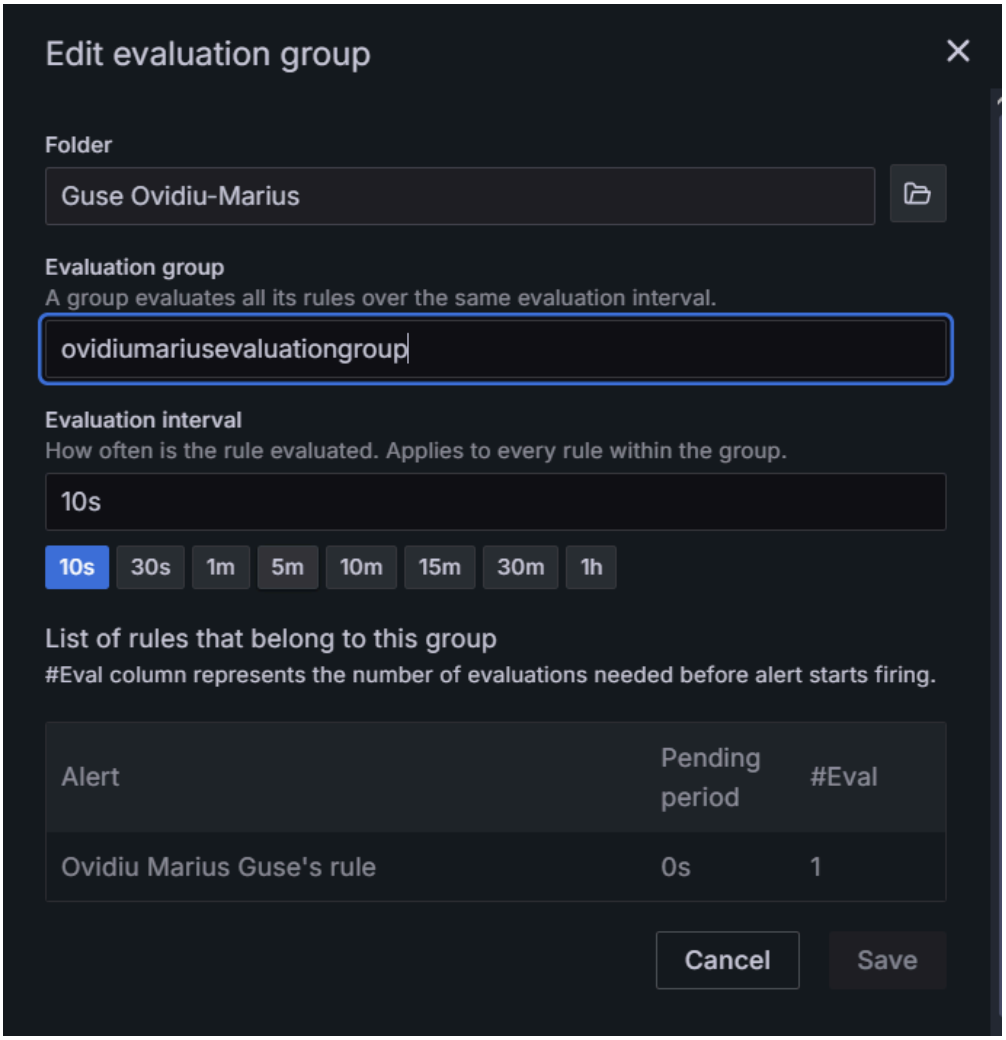
Grafana Alerts:

Grafana's alerting feature allows you to define rules based on the data collected from the data sources. When a specific condition is met (e.g., CPU usage exceeds a certain threshold), Grafana can trigger alerts through various notification channels like email, Slack, or other integrations.

This is the notification channel where I created an alert and tested it with my email:

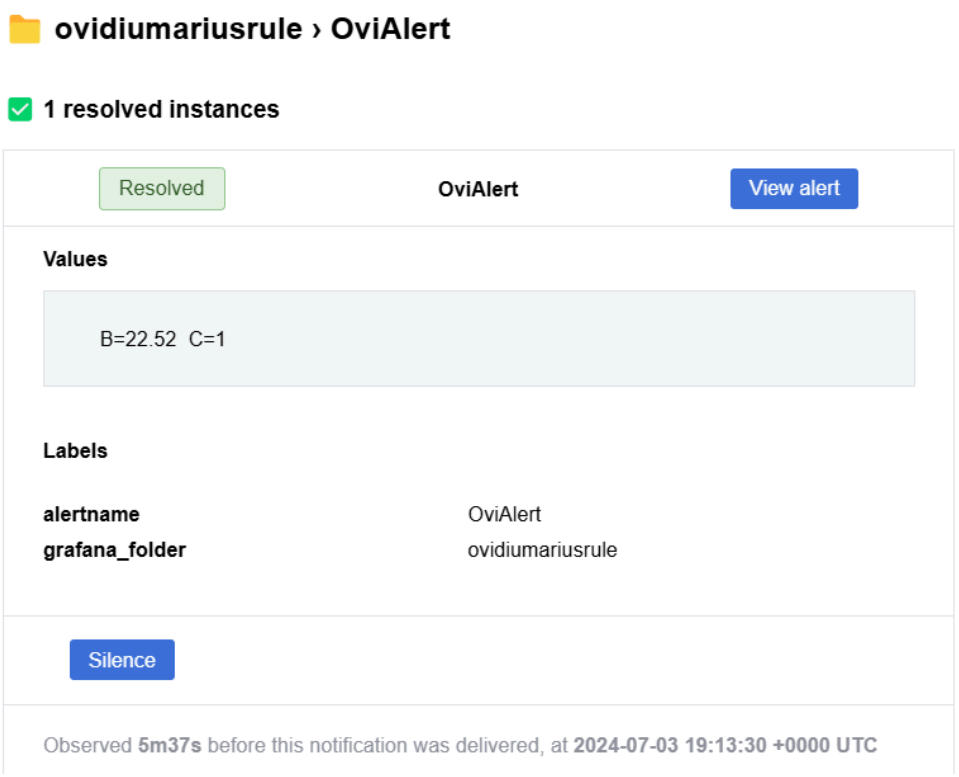


Guse Ovidiu-Marius



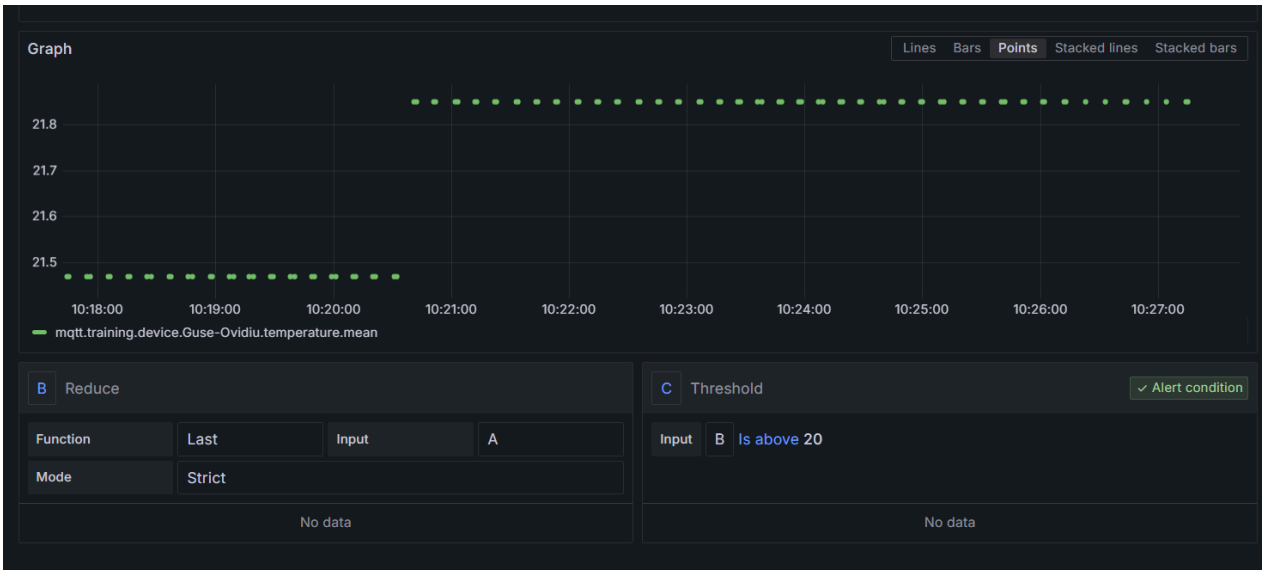
I received on my email the alert that temperature got above a certain number I selected (20):

Guse Ovidiu-Marius



B is the temperature and C is the air pollution

B should’ve been under 20!



There should’ve been a red line showing the threshold after which the alert will be sent to my email, but since the threshold is only 20 and the lowest temperature is already above, the red line cannot be seen, even though it exists!

Guse Ovidiu-Marius

MQTT:

MQTT (Message Query Telemetry Transport) is a lightweight publish-subscribe protocol for machine to machine communication, which can connect devices that are located at remote connections. It offers bi-directional connections, and depending on the requirements for the connection, the communication can be made permanent or it can retrieve data at certain specified intervals, in order to save on bandwidth and on power.
The server we publish the data to is **mqtt.beia-telemetrie.ro**

The topic where the data is published is **training/device/Guse-Ovidiu**

Afterwards, the data is retrieved from the same topic after we have subscribed to it.

Node-Red:

Node-RED is a programming tool for wiring together hardware devices, APIs and online services. It provides a browser-based editor that makes it easy to wire together flows using the wide range of nodes in the palette that can be deployed to its runtime in a single-click.

How it works:

1. Flow-Based Programming: Node-RED adopts a flow-based programming paradigm, where a flow represents a sequence of nodes connected together. Nodes are small blocks of functionality that can be linked to perform specific tasks. The user can create a flow by dragging and dropping nodes onto the workspace and connecting them to define the desired logic.
2. Nodes and Palette: Node-RED offers a palette that contains various pre-built nodes for input, output, processing, and function nodes. The palette can be extended with additional custom nodes created by the community or the user.
    - Input Nodes: Nodes that receive data or trigger events, such as HTTP, MQTT, WebSocket, or sensor nodes.
    - Output Nodes: Nodes that send data or perform actions, such as HTTP response, MQTT publish, email, or notification nodes.
    - Processing Nodes: Nodes that process or transform data, such as JSON, CSV, or function nodes for custom logic.
    - Function Nodes: Custom nodes where users can write JavaScript functions to implement specific operations.
3. Drag-and-Drop Interface: Users can create flows by dragging nodes from the palette onto the workspace and connecting them with wires to define the data flow and logic between nodes.

4. Wiring Nodes: Once nodes are placed on the workspace, wires (representing data connections) can be used to connect the output of one node to the input of another. This enables the flow of data and control between nodes.
5. Event-Driven Execution: Node-RED operates in an event-driven manner. As data arrives at an input node or an event is triggered, the flow is executed sequentially, passing data from node to node along the defined connections.
6. Runtime Execution: Node-RED provides a runtime environment, typically based on Node.js. The flows created using the browser-based editor are deployed to this runtime environment. The runtime manages the execution of the flows and handles events and data flow between nodes.
7. Built-in Dashboard: Node-RED includes a built-in dashboard feature that allows users to create simple web-based user interfaces to control and monitor flows and applications.
8. Integration with IoT and Other Services: Node-RED's vast library of nodes includes integrations with various IoT platforms, cloud services, databases, and other technologies. This makes it easy to interact with a wide range of devices and services within the flow.

Overall, Node-RED's visual and flow-based approach simplifies the development process for connecting and automating tasks, making it a popular choice for rapid prototyping and building IoT applications and automation workflows.

I downloaded and installed Node.js from this website: https://nodejs.org/en/download

Once Node.js was installed, I open the Command Prompt and ran the following command: npm install -g node-red

Once the installation was complete I started Node-Red by running the command “node-red”:

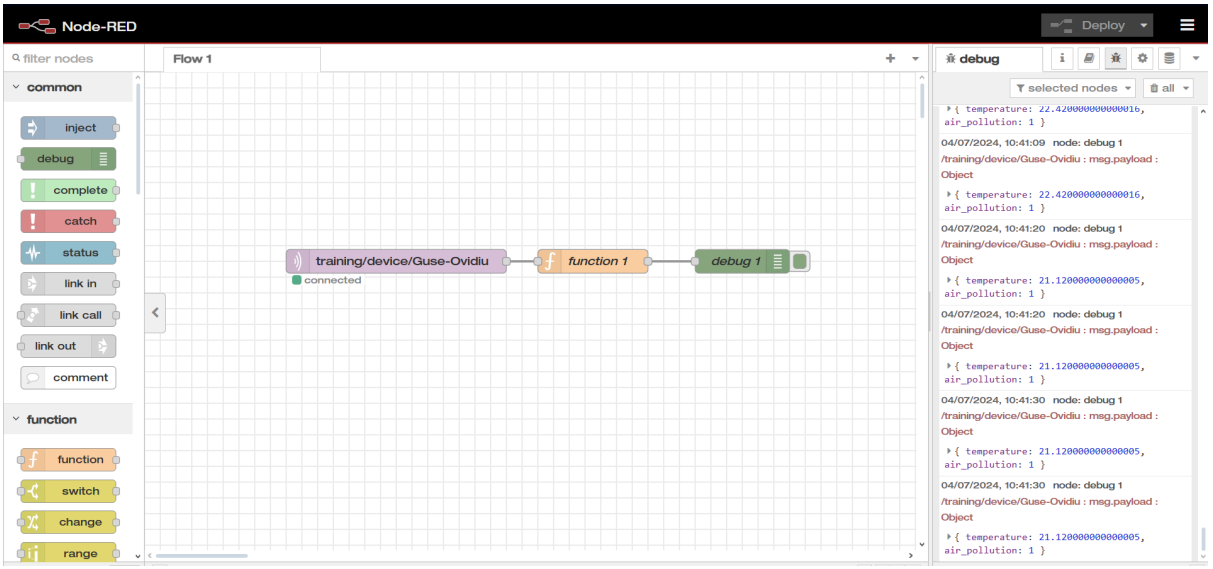
Guse Ovidiu-Marius

```
Welcome to Node-RED
===================

4 Jul 08:07:18 - [info] Node-RED version: v4.0.2
4 Jul 08:07:18 - [info] Node.js  version: v20.15.0
4 Jul 08:07:18 - [info] Windows_NT 10.0.22631 x64 LE
4 Jul 08:07:19 - [info] Loading palette nodes
4 Jul 08:07:25 - [info] Settings file  : C:\Users\Ovi\AppData\Roaming\SPB_Data\.node-red\settings.js
4 Jul 08:07:25 - [info] Context store  : 'default' [module=memory]
4 Jul 08:07:25 - [info] User directory : C:\Users\Ovi\AppData\Roaming\SPB_Data\.node-red
4 Jul 08:07:25 - [warn] Projects disabled : editorTheme.projects.enabled=false
4 Jul 08:07:25 - [info] Flows file     : C:\Users\Ovi\AppData\Roaming\SPB_Data\.node-red\flows.json
4 Jul 08:07:25 - [warn]

---------------------------------------------------------------------
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
---------------------------------------------------------------------

4 Jul 08:07:25 - [info] Server now running at http://127.0.0.1:1880/
4 Jul 08:07:25 - [info] Starting flows
4 Jul 08:07:25 - [info] Started flows
4 Jul 08:07:25 - [info] [mqtt-broker:BEIA] Connected to broker: mqtt://mqtt.beia-telemetrie.ro:1883
```

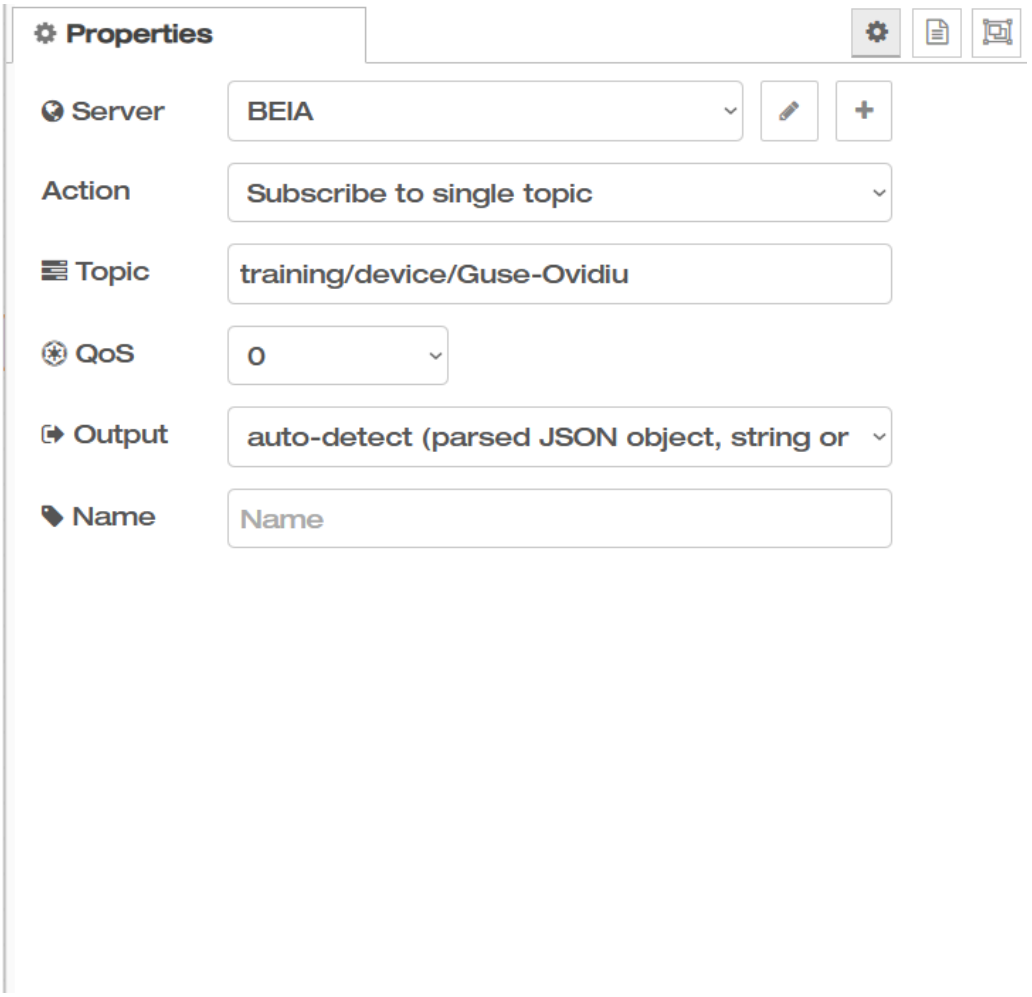
I used Node-RED to connect to an MQTT topic(my MQTT topic that I created) and retrieve values from the broker. I processed these values using a function block and sent the processed data to another MQTT topic.

Then I used a debug block so I can check if the data that is retrieved is the same as the one generated in PyCharm.

Guse Ovidiu-Marius

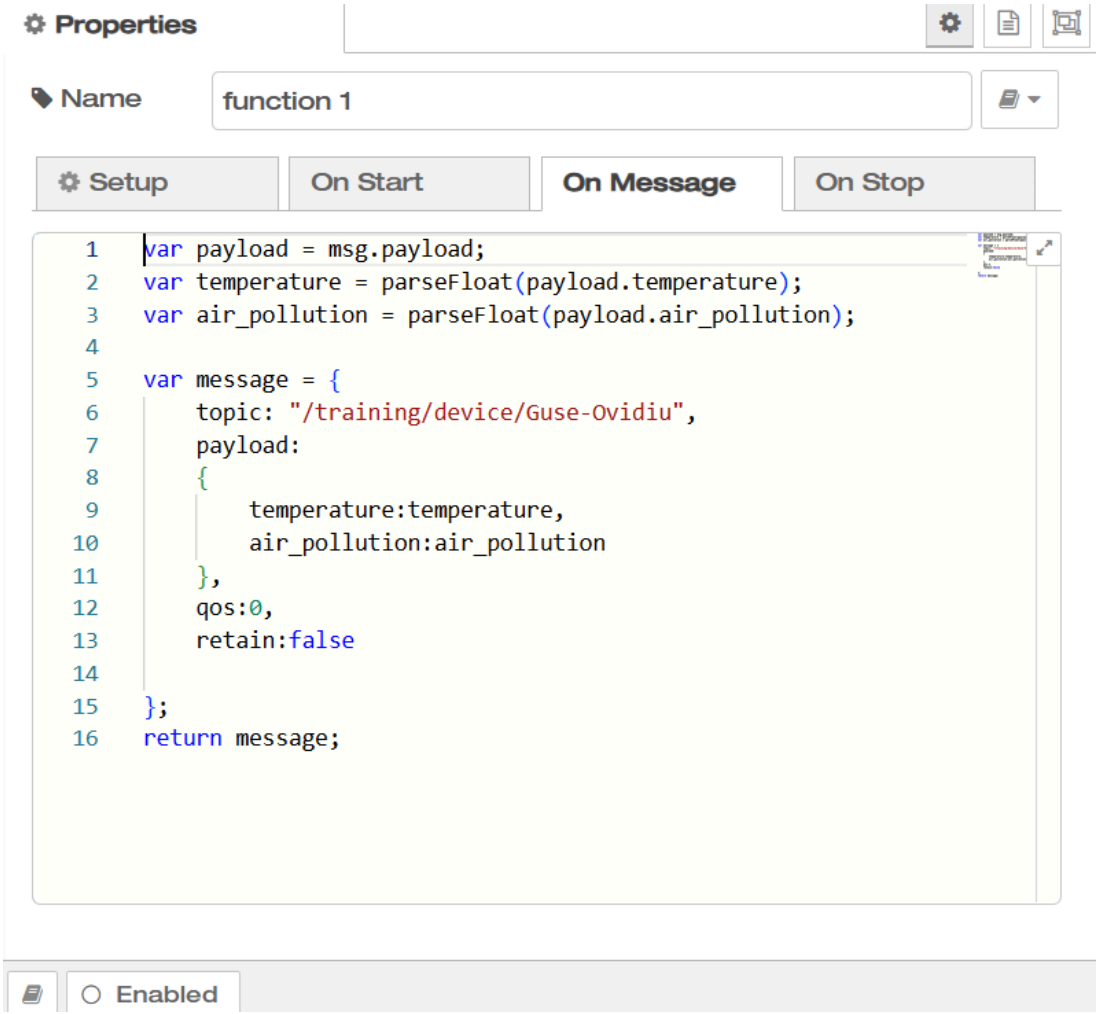


As we can see the temperatures and air pollution number is the same as in PyCharm!



The QoS for the MQTT protocol is set to 0. The board and the broker use the “shoot-and-forget” method for data transmission.

Guse Ovidiu-Marius



The temperature is set to Celsius degrees so it's easier to read and understand!

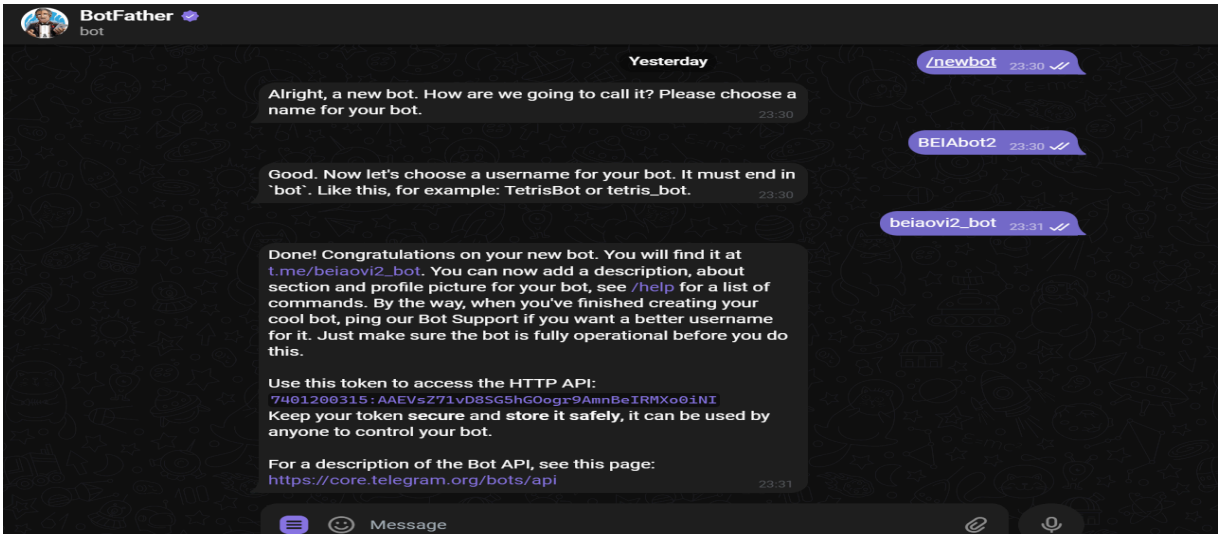
Telegram Chatbot

First of all, I downloaded and installed Telegram on my PC because it's easier than on my smartphone.

Second of all, searched for @BotFather, created a new bot using /newbot command, then gave it a name, and a username.

After this I got the API TOKEN from my bot which I used on my Pycharm code.

Guse Ovidiu-Marius

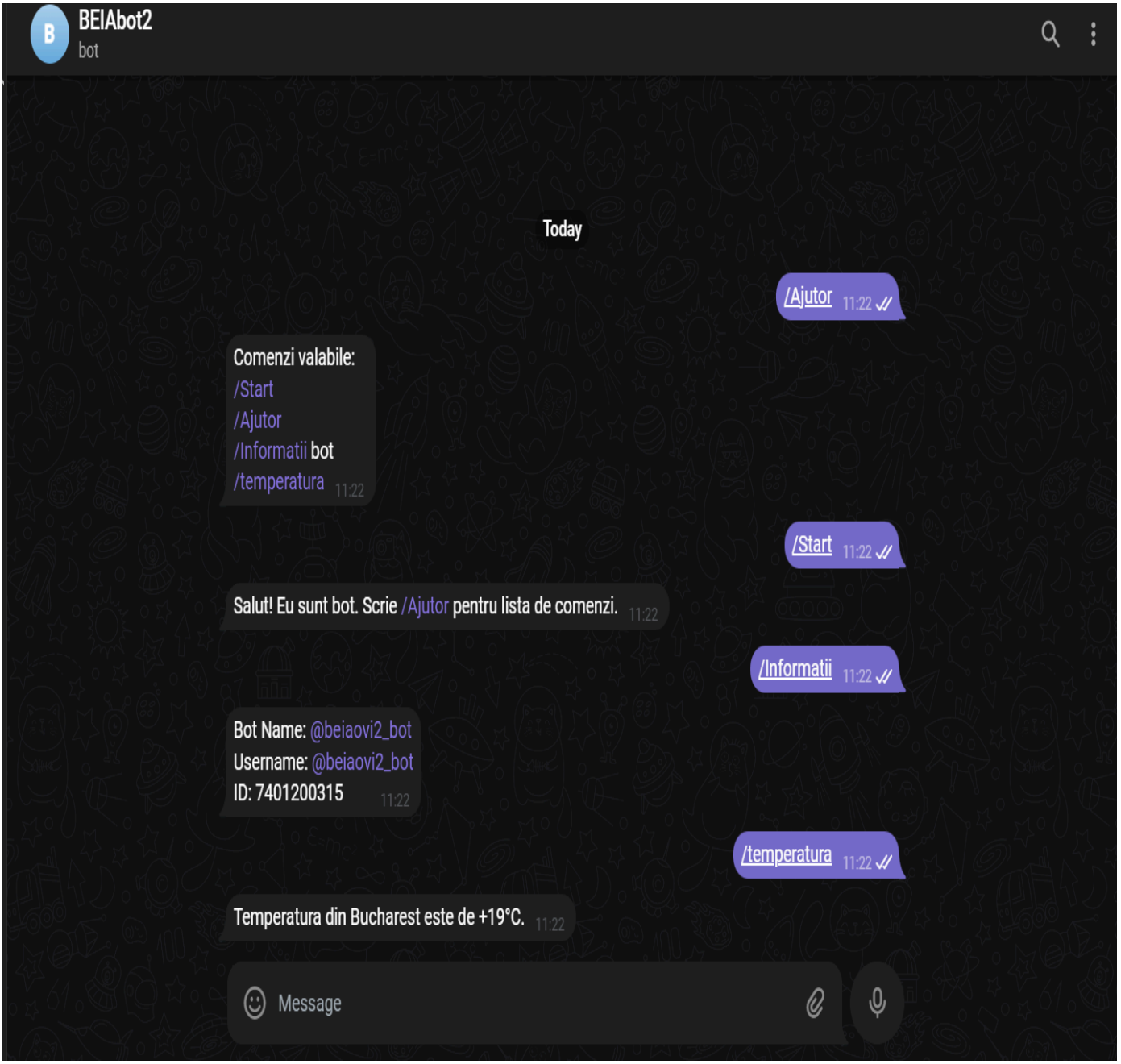


This is the code I used in Pycharm:

```
import requests
from bs4 import BeautifulSoup
from telegram import Update, Bot
from telegram.ext import Application, CommandHandler, CallbackContext
response = requests.get("http://www.google.com")
print(response.status_code)
# Replace the API_KEY_TOKEN with the token provided by BotFather
API_TOKEN = "7401200315:AAEVsZ71vD8SG5hGOogr9AmnBeIRMXo0iNI"
async def start(update: Update, context: CallbackContext):
    await update.message.reply_text("Hello! I am your bot. Send /help to see the list of available commands.")
async def help_command(update: Update, context: CallbackContext):
    help_text = "Available commands:\n"
    help_text += "/start - Start the bot\n"
    help_text += "/help - Show this help message\n"
    help_text += "/info - Get bot information\n"
    help_text += "/temperature - Get the current temperature in Bucharest"
    await update.message.reply_text(help_text)
async def info(update: Update, context: CallbackContext):
    bot_info = "Bot Name: {}\nUsername: @{}\nID: {}".format(
```

```
        context.bot.name, context.bot.username, context.bot.id)
    await update.message.reply_text(bot_info)
async def get_temperature(update: Update, context: CallbackContext) -> None:
    city = "Bucharest"
    url = f"http://wttr.in/{city.replace(' ', '+')}?format=%t"
    response = requests.get(url)
    if response.status_code == 200:
        temperature = response.text.strip()
        await update.message.reply_text(f"The current temperature in {city} is {temperature}.")
    else:
            await update.message.reply_text("Sorry, unable to fetch the temperature at the moment.")
def main():
    application = Application.builder().token(API_TOKEN).build()
    # Add command handlers
    application.add_handler(CommandHandler("start", start))
    application.add_handler(CommandHandler("help", help_command))
    application.add_handler(CommandHandler("info", info))
    application.add_handler(CommandHandler("temperature", get_temperature))
    application.run_polling()
if __name__ == '__main__':
    main()
```

Guse Ovidiu-Marius



Blockchain

Blockchain technology is a revolutionary concept that has gained significant attention in recent years. At its core, a blockchain is a distributed and decentralized digital

ledger that records transactions and data across a network of computers. Unlike traditional centralized databases, blockchain operates without a single point of control, making it inherently transparent, secure, and immutable

A smart contract is a self-executing digital contract with the terms of the agreement written directly into lines of code. It operates on a blockchain platform and automatically enforces the rules and conditions of the contract without the need for intermediaries. Smart contracts enable trust and transparency in business transactions, as they are tamper-resistant, immutable, and executed automatically when predefined conditions are met.

Why should we use blockchain?

1. Enhanced Security: Blockchain's decentralized and cryptographic nature provides a high level of security, making it difficult for unauthorized users to tamper with data or perform fraudulent activities.
2. Transparency and Traceability: The transparency of blockchain allows for easy tracking of transactions and activities, making it particularly useful in supply chain management, auditing, and other industries where provenance is critical.
3. Reduced Intermediaries: By eliminating the need for intermediaries (such as banks or third-party payment processors), blockchain can streamline processes, reduce costs, and increase efficiency.
4. Financial Inclusion: Blockchain technology has the potential to bring financial services to unbanked or underbanked populations, enabling greater financial inclusion worldwide.
5. Smart Contracts: Blockchain platforms like Ethereum allow for the creation of smart contracts, which are self-executing agreements with predefined rules. Smart contracts can automate processes, eliminate middlemen, and reduce the risk of contract breaches.
6. Data Integrity: Blockchain's immutability ensures that once data is recorded, it cannot be altered retroactively. This feature is valuable in various applications, including legal documentation and intellectual property rights management.

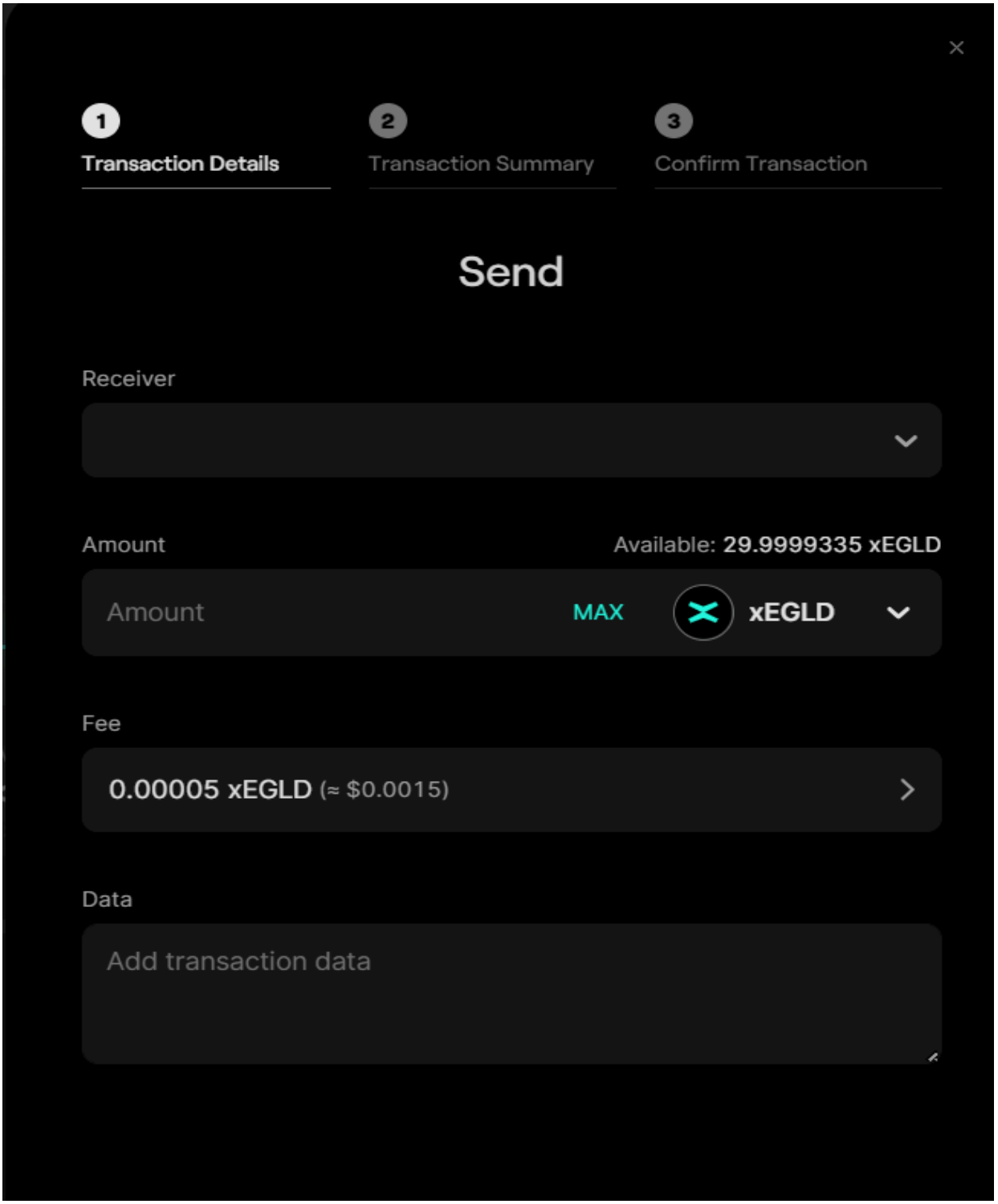
I created a blockchain using the website https://testnet-wallet.multiversx.com/unlock

I made some transactions by sending some XEGLD

And the data sent with a transaction is the data from the sensor which I can also automate, example:

If temperature gets above a level, I could automatically do a transaction!

Guse Ovidiu-Marius



Guse Ovidiu-Marius

Docker & ArrowHead

„Docker is a set of platform as a service (PaaS) products that uses OS-level virtualization to deliver software in packages called containers. The software that hosts the containers is called Docker engine.

A container is a lightweight, standalone, and executable software package that includes everything needed to run a piece of software, including the code, runtime, system tools, system libraries, and settings. Containers provide a consistent and isolated environment for applications to run across different computing environments, such as development, testing, and production.”

1.Installed Docker Desktop

2.Created dockerfile in my Docker Folder to define Arrowhead image.

3.Next I created the Dockerfile image.

4.I launched the Docker Image

Docker run -p 3001:3000 -d ovid1

5.Made sure I have all the files:

`Dockerfile`

`Package.json`

`Package-lock.json`

6.Command to create Arrow image

Sudo odcker build -t arrow .

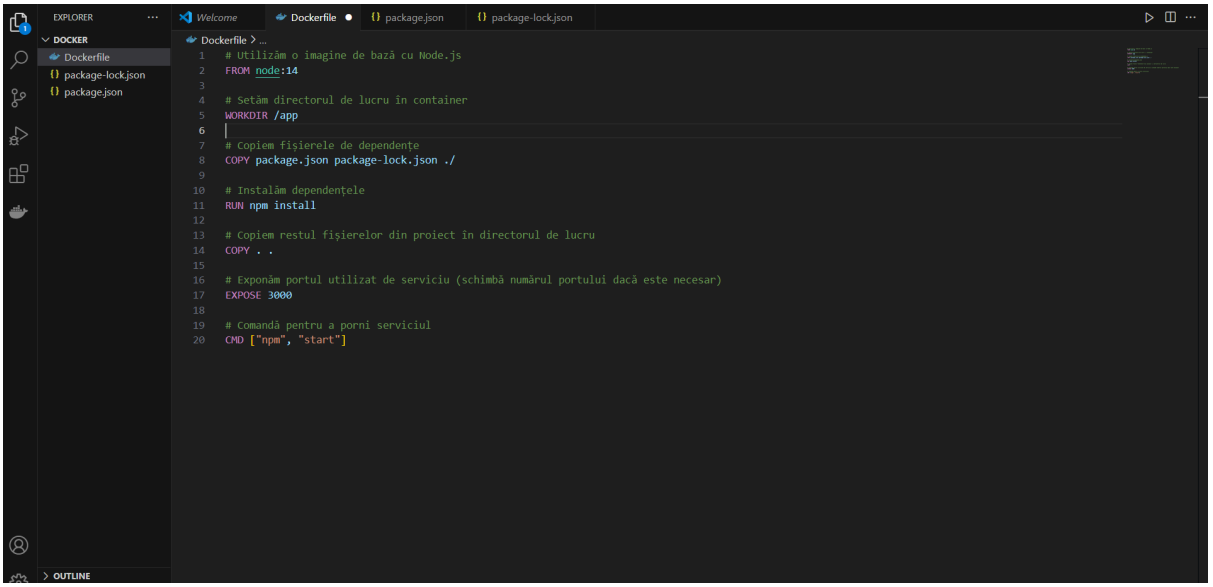
Package.json:

```
{
  "name": "my-app",
  "version": "1.0.0",
  "description": "A simple Node.js application",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

Guse Ovidiu-Marius

Index.js:

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
  res.send('Hello, world!');
});

app.listen(port, () => {
  console.log(`App running at http://localhost:${port}`);
});
```



Guse Ovidiu-Marius