

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Лабораторна робота №9

з дисципліни «Інженерія систем IoT»

Тема: «Захист IoT застосунків»

Варіант: 65

Виконав:

студент групи IA-33

Заранік Максим

Перевірив:

асистент кафедри ICT

Головатенко І. А.

Київ 2025

Мета: Мета: У кількох останніх лабораторних роботах ви створили IoT-пристрій моніторингу ґрунту та підключили його до хмари. Але що, якщо хакерам, які працюють на фермера-конкурента, вдалося захопити контроль над вашими пристроями IoT? Що, якби вони надіслали дані про високу вологість ґрунту, щоб ваші рослини ніколи не поливали, або ввімкнули вашу систему поливу, щоб вона постійно працювала, вбиваючи ваші рослини через надмірний полив? У цій лабораторній роботі ви дізнаєтесь про захист пристройів Інтернету речей. Ви також дізнаєтесь, як очистити свої хмарні ресурси, зменшивши по тенційні витрати.

Хід роботи

Хід роботи

1. Створено новий IoT-пристрій із сертифікатом X.509.
За допомогою команди `az iot hub device-identity create` згенеровано пару ключів RSA та самопідписаний сертифікат X.509. У поточному каталогі створено два файли: приватний ключ та сертифікат пристрою.
1. Скопійовано сертифікат і ключ до каталогу IoT-пристрою.
Файли `soil-moisture-sensor-x509-cert.pem` та `soil-moisture-sensor-x509-key.pem` перенесено в папку з програмою пристрою.
2. Модифіковано код пристрою для підключення через X.509.
У файл `app.py` додано змінні `host_name` та `device_id`, імпортовано клас X509, створено об'єкт сертифіката та змінено ініціалізацію клієнта на:
`IoTHubDeviceClient.create_from_x509_certificate(...)`.
Рядок підключення видалено.
3. Перезапущено програму пристрою та виконано тестування.
Пристрій успішно підключився до IoT Hub із використанням сертифіката X.509, продовжив надсилати телеметрію та приймати прямі методи керування.

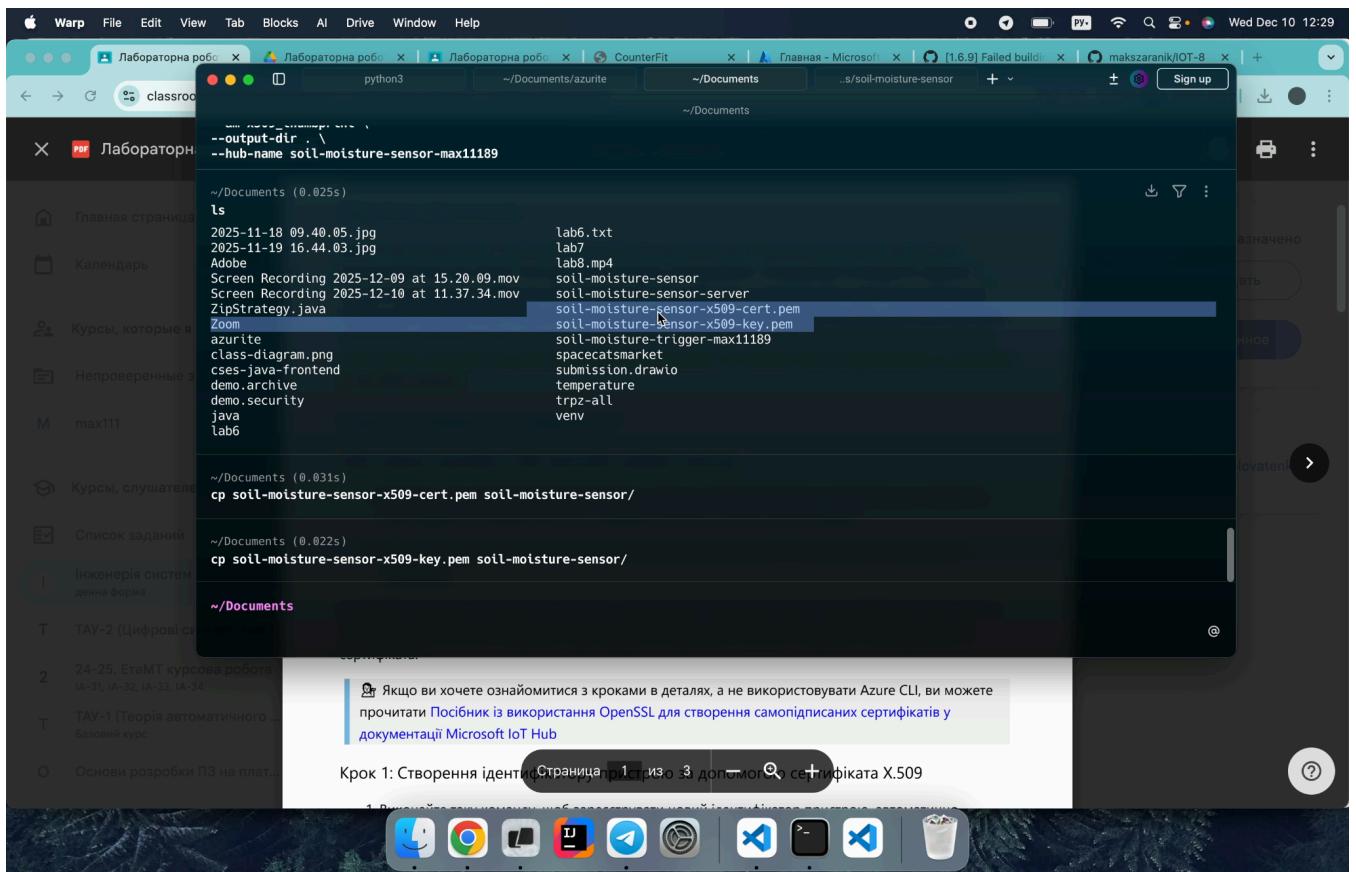


Рисунок 1.1 - Генерація та копіювання сертифікатів X509.

Код клієнтської частини:

```
app.py M X
app.py
1 import json
2 import time
3 import paho.mqtt.client as mqtt
4 import threading
5
6 id = '9e9e2074-64b1-4dd2-8ee3-ad2ab0359a77'
7 client_telemetry_topic = id + '/telemetry'
8 server_command_topic = id + '/command'
9 client_name = id + 'soil_moisture_sensor_server'
10
11 mqtt_client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2, client_name)
12 mqtt_client.connect('test.mosquitto.org')
13 mqtt_client.loop_start()
14
15 water_time = 5
16 wait_time = 20
17
18 def send_relay_command(client, state):
19     command = {'relay_on': state}
20     print("Sending message:", command)
21     client.publish(server_command_topic, json.dumps(command))
22
23 def control_relay(client):
24     mqtt_client.unsubscribe(client_telemetry_topic)
25     send_relay_command(client, True)
26     time.sleep(water_time)
27     send_relay_command(client, False)
28     time.sleep(wait_time)
29     mqtt_client.subscribe(client_telemetry_topic)
30
31 def handle_telemetry(client, userdata, message):
32     payload = json.loads(message.payload.decode())
33     print("Message received:", payload)
34     if payload['soil_moisture'] > 281:
35         threading.Thread(target=control_relay, args=(client,)).start()
36
37 mqtt_client.subscribe(client_telemetry_topic)
38 mqtt_client.on_message = handle_telemetry
39
40 while True:
41     time.sleep(2)
```

Код функції:

The screenshot shows a Microsoft Visual Studio Code interface running on a Mac. The window title is "soil-moisture-sensor". The left sidebar shows a project structure with a ".venv" folder and files "app.py", "soil-moisture-sensor-x509-cert.pem", and "soil-moisture-sensor-x509-key.pem". The main editor area displays the "app.py" file content:

```
app.py
1  from counterfit_connection import CounterFitConnection
2  import time
3  from counterfit_shims_grove.adc import ADC
4  from counterfit_shims_grove.grove_relay import GroveRelay
5  import json
6  from azure.iot.device import IoTHubDeviceClient, Message, MethodResponse
7  from azure.iot.device import IoTHubDeviceClient, Message, MethodResponse, X509
8
9  host_name = "soil-moisture-sensor-max11189.azure-devices.net"
10 x509 = X509("./soil-moisture-sensor-x509-cert.pem", "./soil-moisture-sensor-x509-key.pem")
11 device_id = "soil-moisture-sensor-x509"
12
13 device_client = IoTHubDeviceClient.create_from_x509_certificate(x509, host_name, device_id)
14 print('Connecting')
15 device_client.connect()
16 print('Connected')
17
18 CounterFitConnection.init('127.0.0.1', 5001)
19 adc = ADC()
20 relay = GroveRelay(66)
21
22
23 def handle_command(client, userdata, message):
24     payload = json.loads(message.payload.decode())
25
max@max-MacBook-Pro soil-moisture-sensor % python app.py
Soil moisture: 250
Sending telemetry: {"soil_moisture": 250}
Direct method received - relay_on
-> Turning relay ON
Soil moisture: 238
Sending telemetry: {"soil_moisture": 238}
Direct method received - relay_off
-> Turning relay OFF
Soil moisture: 270
Sending telemetry: {"soil_moisture": 270}
Soil moisture: 233
Sending telemetry: {"soil_moisture": 233}
Exception caught in background thread. Unable to handle.
['azure.iot.device.common.transport_exceptions.ConnectionDroppedError: Unexpected discor
-> Turning relay OFF
[]
```

The terminal output shows the application running and sending telemetry data. A tooltip in the bottom right corner asks if the user wants to install the recommended 'Python' extension from Microsoft. The status bar at the bottom indicates the user is signed out.

Рисунок 1.1 - Локальна конфігурація для пристрою.

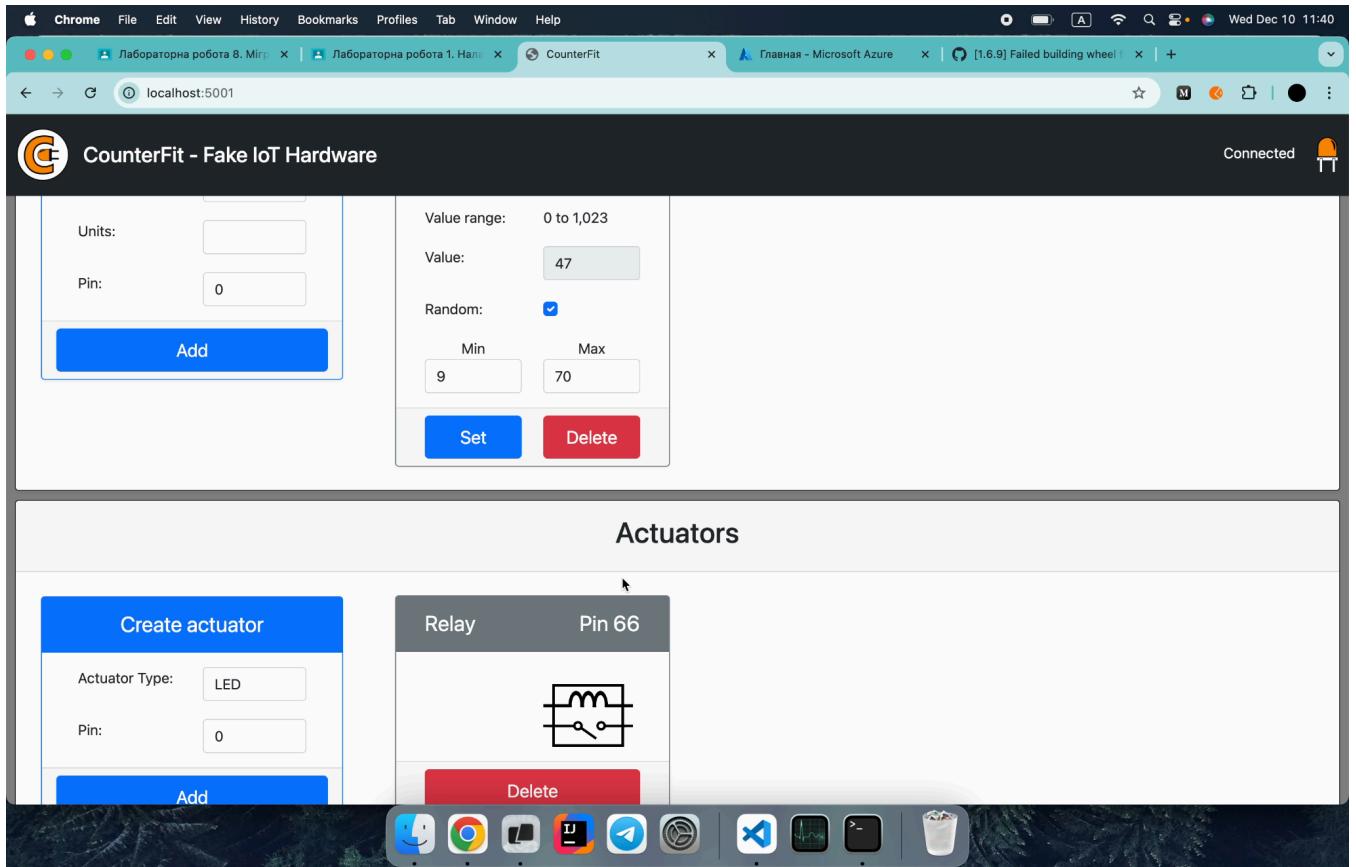


Рисунок 1.2 - Інтерфейс CounterFit із сенсором вологості та актуатором Relay.

The screenshot shows the Visual Studio Code (VS Code) interface. The left sidebar shows a project structure with files like 'function_app.py', 'host.json', 'local.settings.json', and 'requirements.txt'. The main editor area displays the 'function_app.py' file, which contains Python code for an Azure Function. The code uses the Azure IoT Hub SDK to handle events from an Event Hub and trigger direct methods on a relay device. The bottom of the screen shows the 'OUTPUT' tab, which displays the execution logs of the function. The logs show messages being received from an event hub and triggered methods being sent to a relay device.

```

import json
import os
from azure.iot.hub import IoTHubRegistryManager
from azure.iot.hub.models import CloudToDeviceMethod

app = func.FunctionApp()

@app.event_hub_message_trigger(arg_name="azeventhub",
                                event_hub_name="iothub-ehub-soil-moist-66140085-c8e8ee0eee",
                                connection="EventHubConnectionString")
def eventhub_trigger(azeventhub: func.EventHubEvent):
    body = json.loads(azeventhub.get_body()).decode('utf-8'))
    device_id = azeventhub.iothub_metadata['connection-device-id']
    logging.info(f'Received message: {body} from {device_id}')

    soil_moisture = body['soil_moisture']
    if soil_moisture > 49:
        direct_method = CloudToDeviceMethod(method_name='relay_on', payload='{}')
    else:
        direct_method = CloudToDeviceMethod(method_name='relay_off', payload='{}')

    logging.info(f'Sending direct method request for {direct_method.method_name} for device {device_id}')

    registry_manager_connection_string = os.environ['REGISTRY_MANAGER_CONNECTION_STRING']
    registry_manager = IoTHubRegistryManager(registry_manager_connection_string)
    registry_manager.invoke_device_method(device_id, direct_method)

    logging.info('Direct method request sent!')

```

Рисунок 1.3 - Редагування function_app.py у VS Code та відображення логів виконання

direct-method.

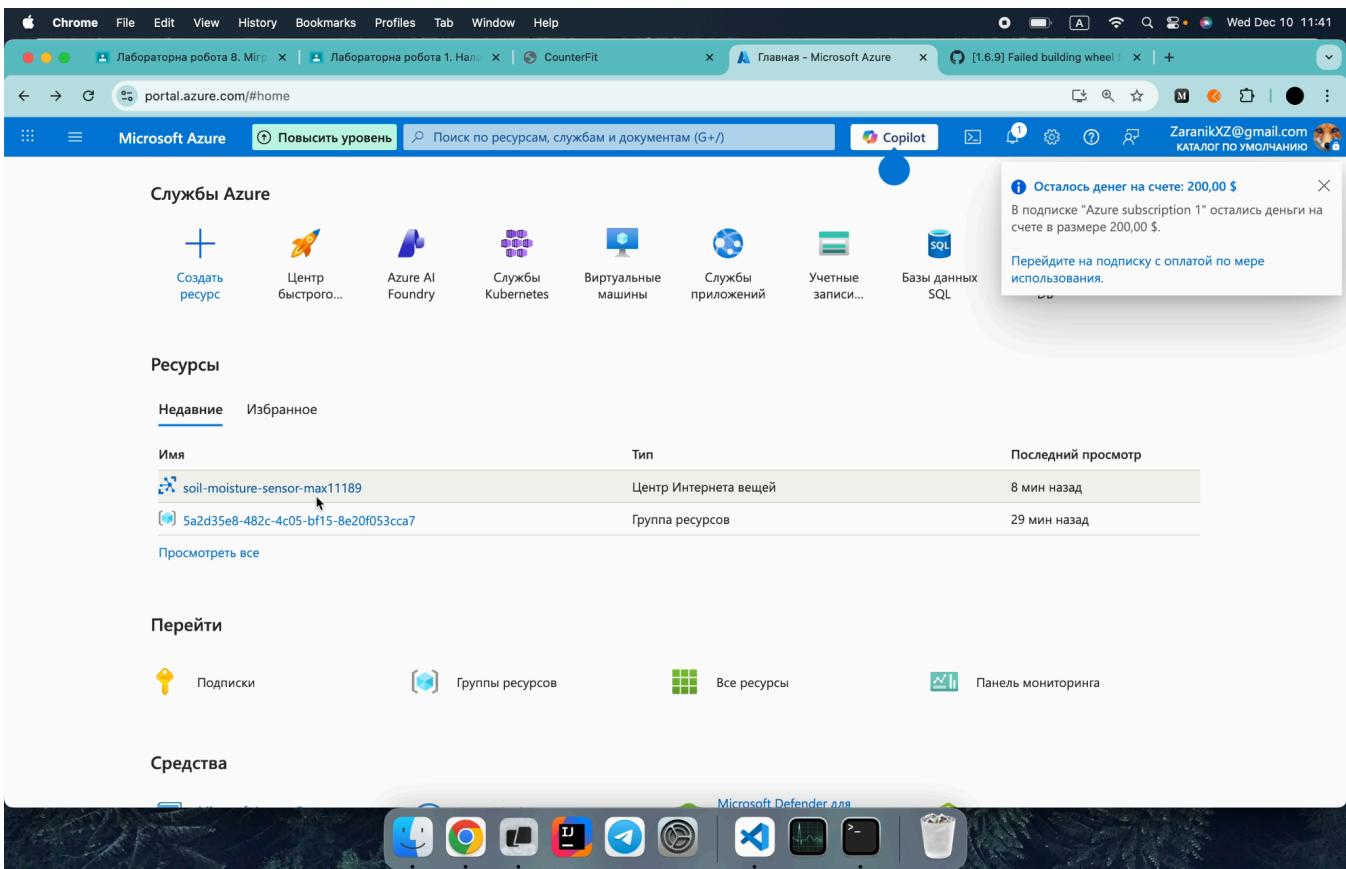


Рисунок 1.4 - Панель Azure Portal із відкритим ресурсом IoT Hub.

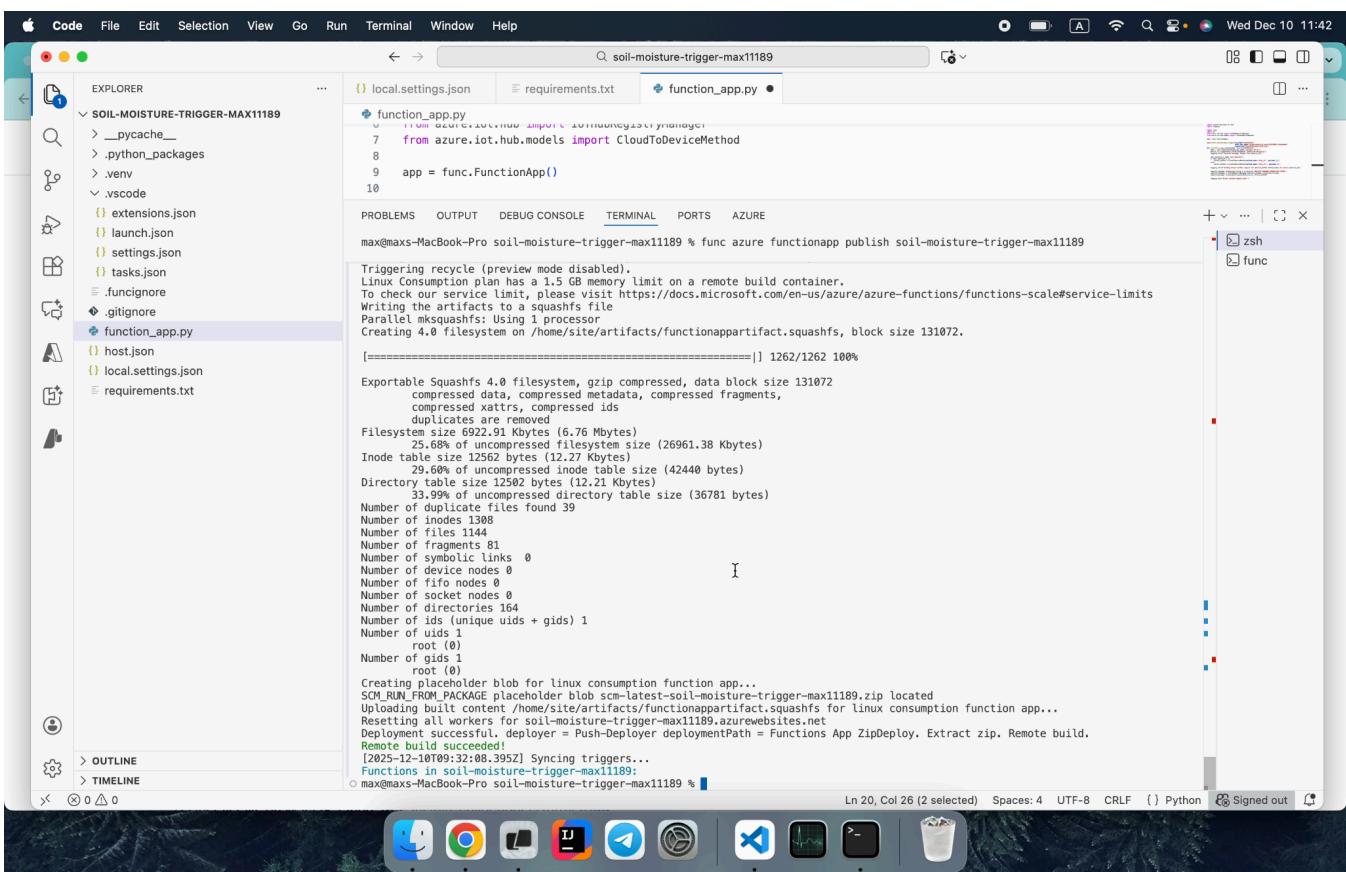


Рисунок 1.5 - Процес публікації Azure Function у терміналі та повідомлення про невідповідність Python-версії.

2. Результат роботи:

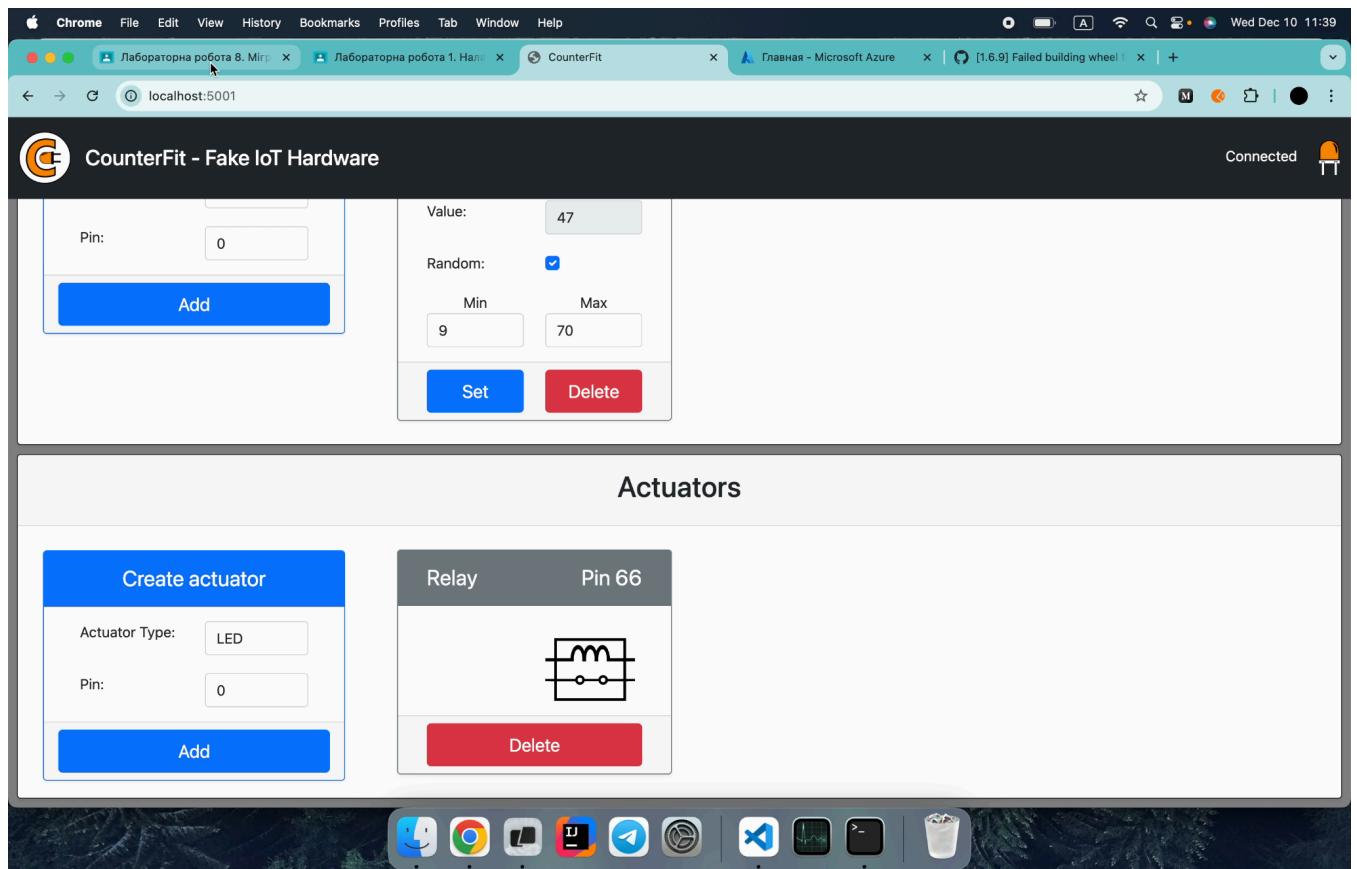


Рисунок 2.1 - інтерфейс Counterfit

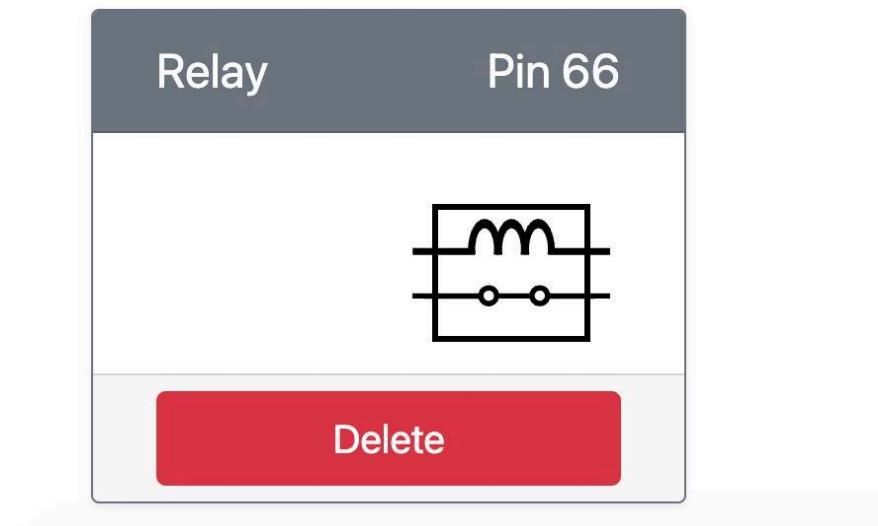


Рисунок 2.2 - стан реле “замкнутий”

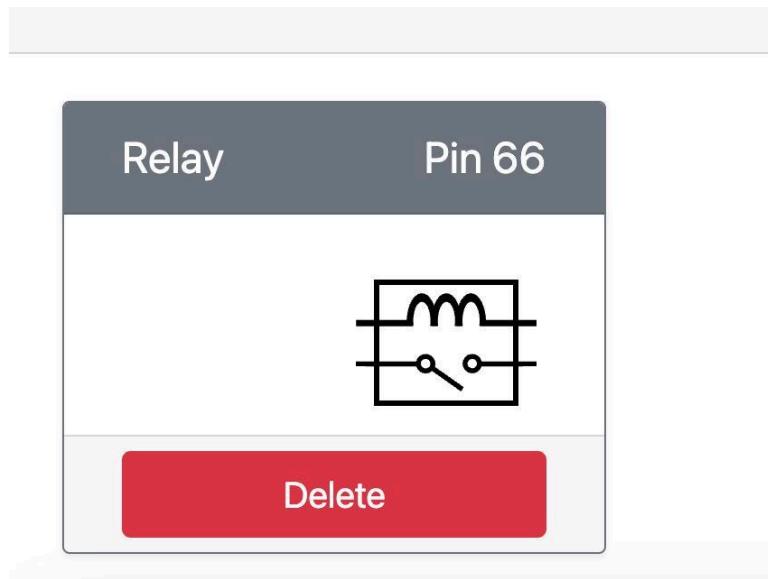


Рисунок 2.3 - стан реле “розімкнущий”

Посилання на github: <https://github.com/makszaranik/IOT-9>

Висновок:

У ході виконання лабораторної роботи №9 було опановано практичні підходи до забезпечення безпеки IoT-рішень шляхом використання сертифікатів X.509 замість традиційних рядків підключення. У межах завдання було створено новий ідентифікатор пристрою в Azure IoT Hub, автоматично згенеровано пару відкритого та приватного ключів, а також самопідписаний сертифікат. Ці ключові артефакти були інтегровані у програмний код IoT-пристрою, що дало змогу встановлювати автентичне та захищене підключення до хмари.

Під час практичного налаштування було модифіковано код пристрою для створення клієнта за допомогою X.509-сертифікату, а також проведено тестування коректності передавання телеметрії та обробки прямих викликів методів (Direct Methods). У результаті підтверджено, що IoT-пристрій успішно автентифікується у хмарі без використання секретних ключів у відкритому вигляді, що значно підвищує загальний рівень безпеки системи.

Отримані навички дозволили зрозуміти принципи побудови захищених IoT-інфраструктур, важливість використання сертифікатів та механізмів криптографічної автентифікації, а також освоїти практичні інструменти Azure для керування ідентичністю пристройів і контролю доступу.