

Smart IoT Parking

Zaharia Eric-Alexandru, 345C1

Cuprins:

- Introducere
- Arhitectura
- Implementare
- Vizualizare si Procesare de Date
- Securitate
- Provocări și Soluții
- Instructiuni de rulare

Sistem de monitorizare parcare

1. Introducere

1. Descriere generala

Acest proiect implementează un sistem inteligent de monitorizare a parării care utilizează tehnologia IoT pentru a oferi actualizări în timp real privind disponibilitatea locurilor de parcare și pentru a automatiza controlul barierelor pe baza detectării vehiculelor.

2. Obiectivele proiectului:

- Detectarea în timp real a ocupării locurilor de parcare cu ajutorul senzorilor de proximitate.
- Control automat al barierelor de parcare pe baza datelor de disponibilitate.
- Vizualizarea și monitorizarea datelor folosind Kibana pentru luarea deciziilor și analiză.

3. Tehnologii folosite:

- Senzori: Senzori cu ultrasunete (HCSR04).
- Actuatori: Servomotor (pentru bariera).
- Comunicare: protocol MQTT.
- Vizualizarea datelor: Elastic Search si Kibana.

2. **Arhitectura**

1. **Senzori (Senzori de proximitate):**

- Tip: senzori ultrasonici HCSR04.
- Măsoară distanța pentru a detecta dacă locurile de parcare sunt ocupate.

2. **Actuator (servomotor):**

- Tip: Servomotor.
- Deschide sau închide bariera în funcție de disponibilitatea locurilor de parcare.

3. **Aplicație de control:**

- Rulează pe Raspberry Pi Pico WH (MicroPython).
- Procesează datele senzorului, comunică cu MQTT și trimite comenzi la actuator.

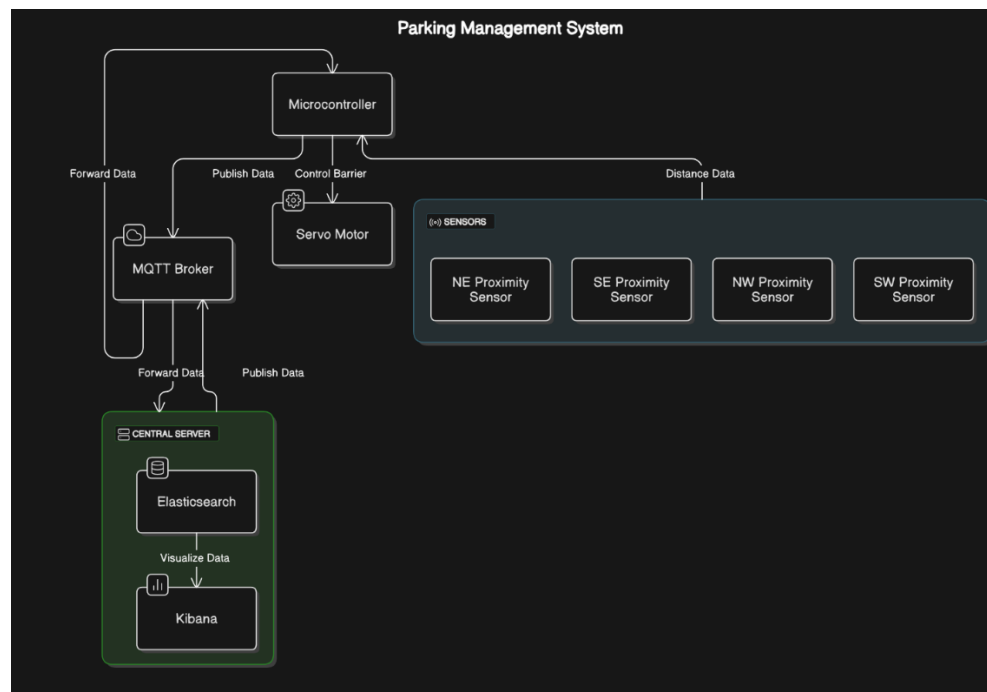
4. **Broker MQTT:**

- Rulează folosind Mosquitto.
- Facilitează comunicarea între microcontroller și serverul central.
-

5. **Server central:**

- Procesează datele citite din broker și comandă închiderea/deschiderea barierei în funcție de aceste date.
- Trimite datele de la senzori procesate către Elasticsearch.
- Software: Elasticsearch și Kibana.
- Elasticsearch stochează datele de parcare pentru interogare, Kibana ajută la vizualizarea tendințelor și ocupării în timp real.

Diagrama componentelor aplicatiei



3. Implementare

Configurarea hardware-ului:

1. Senzori cu ultrasunete HCSR04:

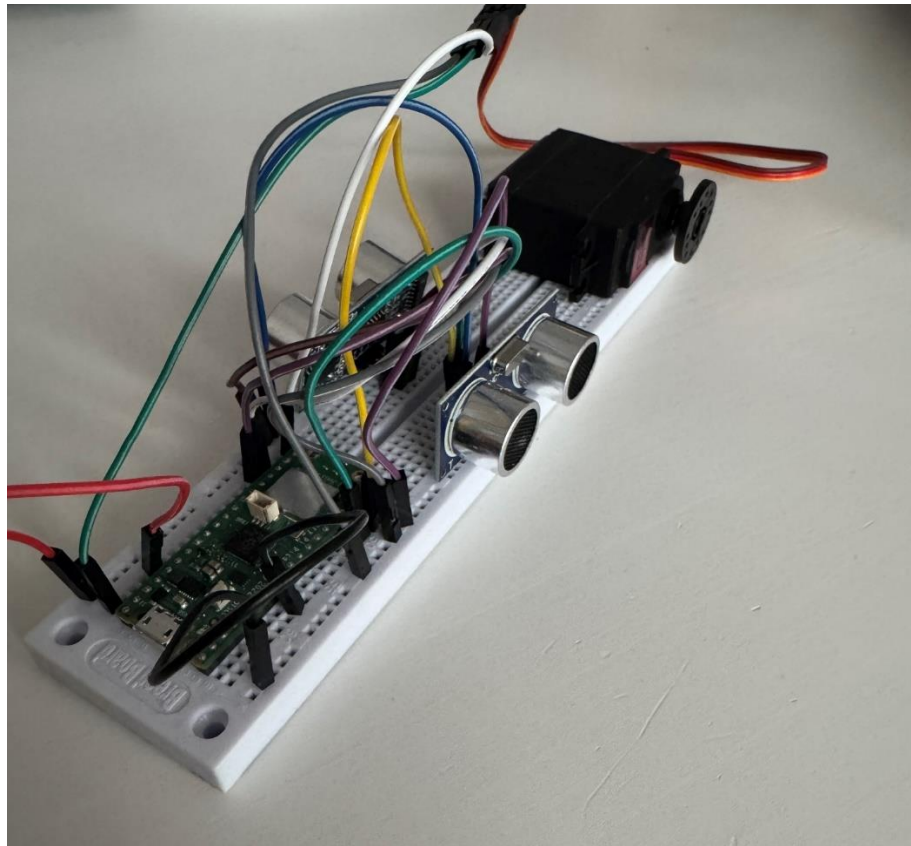
- Am montat senzorii in pozitiile corespunzatoare pentru detectarea vehiculelor.
- Am conectat pinul de **Trigger** si **Echo** al fiecarui senzor la porturile GPIO desemnate pe Raspberry Pi Pico WH.
- Am alimentat senzorii de la sursa de 3V a microcontrolerului.

2. Servomotor:

- Alimentez servomotor-ul la pin-ul VBUS.
- Am conectat PWM la un port compatibil al placutei.

3. Microcontroller:

- Am conectat Raspberry Pi Pico W la alimentare si i-am configurat conexiunea Wi-Fi introducand SSID-ul si parola in script-ul scris in MicroPython.



Configurarea software-ului:

1. Am instalat dependintele necesare pentru Raspberry Pi Pico W:
 - MicroPython: Am incarcat firmware-ul compatibil pe placa.
 - Biblioteci Python: Am adaugat librariile hcsr04, machine, network si paho.mqtt in proiect.
2. Am configurat MQTT:
 - Am instalat si rulat Mosquitto MQTT Broker pe server.
 - Am conectat microcontroller-ul la broker prin portul 1883.
 - Microcontroller-ul publica date despre senzori si citeste date despre decizia server-ului privind starea barierei (0 sau 1), apoi actioneaza servomotorul.

```
def on_message(topic, msg):  
    if msg.decode() == "1":  
        set_servo_angle(0)  
        message = "Bariera inchisa"  
    else:  
        set_servo_angle(90)  
        message = "Bariera deschisa"  
    print(message)
```

```
while True:  
    distance_e = sensor_e.distance_cm()  
    distance_w = sensor_w.distance_cm()  
    message = f"E-{{abs(distance_e)}},W-{{abs(distance_w)}}"  
    client.publish(topic, message)  
  
    client.check_msg()  
    sleep(1)
```

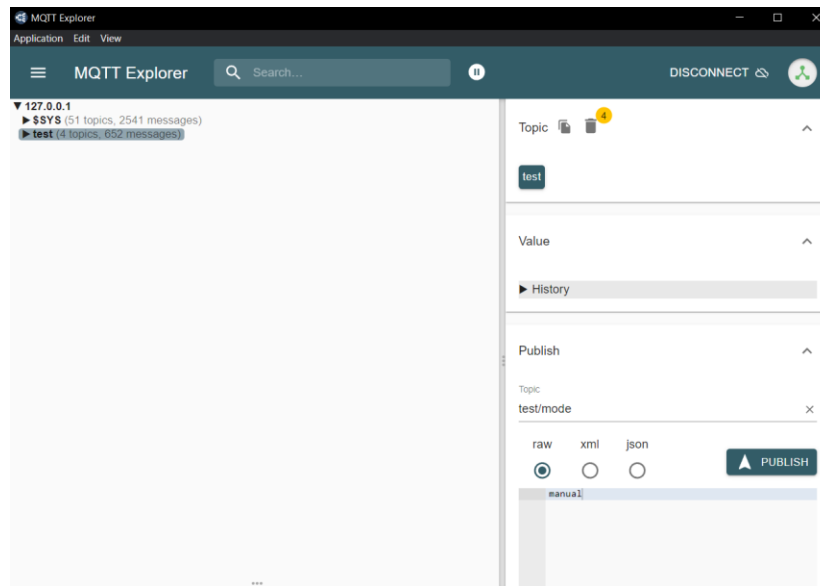
- Am conectat client-ul de Python la broker, cu scopul de a citi datele si a le adauga in Elasticsearch.

```
es = Elasticsearch("https://localhost:9200",  
                   basic_auth=("elastic", "hoG9ZsOilolQQiGMxKX2"),  
                   verify_certs=True,  
                   ca_certs="certs/ca.crt")
```

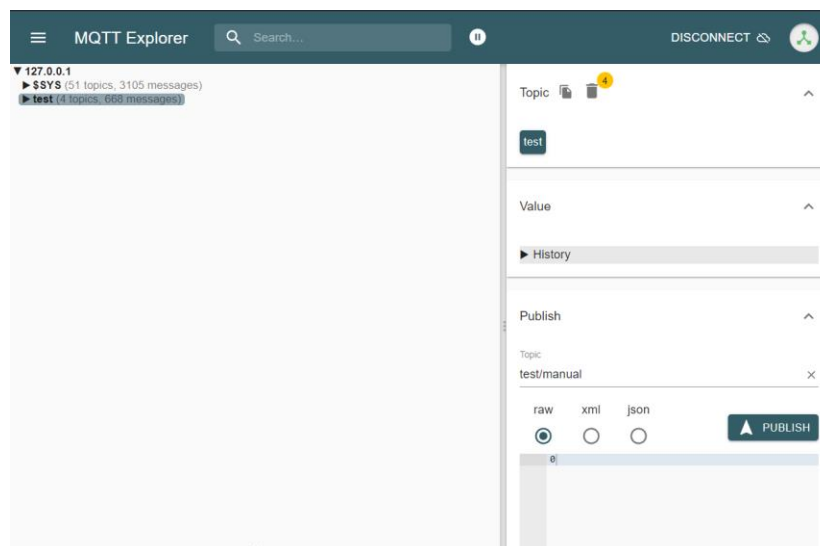
Server-ul de Python proceseaza datele, calculand daca parcareea este plina (si bariera inchisa pentru a bloca accesul masinilor). Daca este plina, trimitem 1 si bariera se inchide, altfel trimitem 0.

```
def on_message(client, userdata, msg):  
    occupied_slots = {}  
    sensors = msg.payload.decode().split(",")  
  
    for sensor in sensors:  
        sensor_id, distance = sensor.split("-")  
        distance = float(distance)  
        occupied_slots[sensor_id] = distance < 10  
  
    data = {}  
    data["sensors"] = occupied_slots.copy()  
    data["timestamp"] = datetime.datetime.now(bucharest_tz).isoformat()  
    es.index(index="parking_data", body=data)  
  
    parking_full = all(data["sensors"].values())  
    client.publish("test/barrier", "1" if parking_full else "0")
```


- Aplicatia suporta activarea modului manual, folosind MQTT Explorer pentru interfata grafica



- Daca suntem in modul manual, trimitem pe topic-ul test/manual 0 (pentru deschiderea barierei) sau 1 (pentru inchiderea ei).



3. Am configurat ElasticSearch si Kibana:

- Am pornit containerele folosind configuratia docker-compose.
- Am creat indexul parking_data in ElasticSearch pentru stocarea datelor.
- In Kibana, am configurat un dashboard pentru vizualizarea datelor.
- Am implementat sistemul de alertare tot in Kibana.

Alertele sunt afisate in server-logul din Kibana, deoarece integrarea cu email nu este inclusa ca feature gratuit al stack-ului ELK.

```
kibana | [2025-01-10T00:42:08.514+00:00][INFO ][plugins.actions.server-log] Server log: Rule 'Empty Parking Slot Log Alert' is active for group 'all documents':;;- Value: 2;- Conditions Met: count is greater than 0 over 5m;- Timestamp: 2025-01-10T00:42:05.686Z
```

Regula de alertare din Kibana:

×

Edit rule

Index threshold

Alert when an aggregated query meets the threshold. [Learn more](#)

Select an index

INDEX parking_data

WHEN count()

OVER all documents

Define the condition

IS ABOVE 0

FOR THE LAST 5 minutes

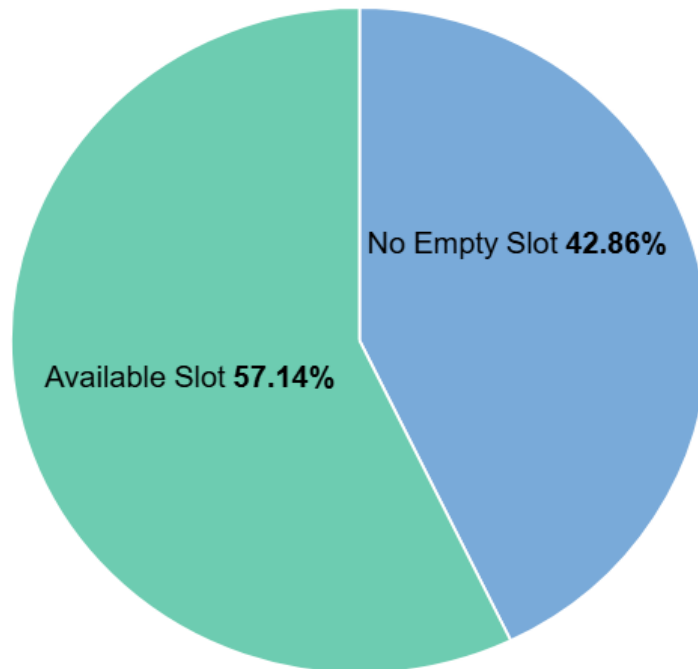
4. Vizualizare si Procesare de Date

Metoda de procesare a datelor:

1. Microcontroller-ul publica date despre distantele masurate de senzori pe topicul test/parking-data.
2. Scriptul de pe server (es_proxy.py) proceseaza aceste date:
 - Compara distantele masurate cu pragul definit (ex.: <10 cm pentru loc ocupat).
 - Genereaza un dictionar care indica starea fiecarui loc de parcare.
 - Trimite datele procesate catre Elasticsearch pentru stocare si analiza ulterioara.
 - Trimite pe MQTT date despre ocuparea parcarii.

Vizualizare in Kibana:

1. Am creat un dashboard in Kibana pentru monitorizarea locurilor de parcare:
 - Am adaugat un grafic pie chart care arata procentul de timp in care locurile sunt ocupate versus momentele in care exista locuri libere.

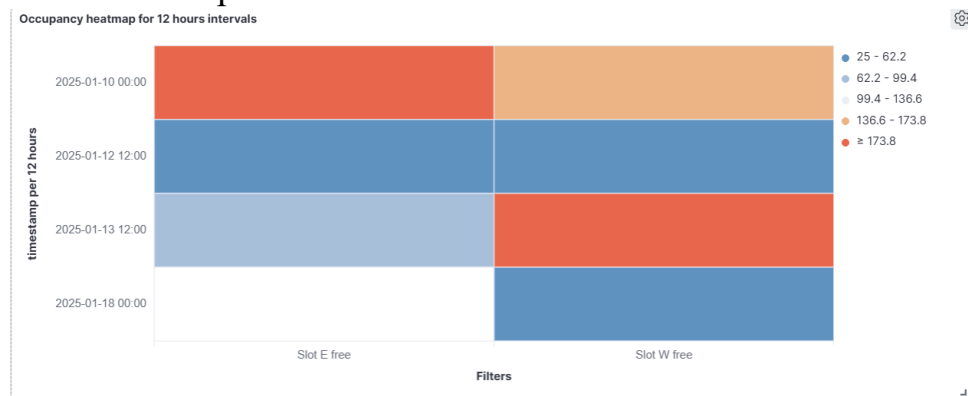


- Am inclus un timeline pentru a vizualiza schimbarile de ocupare in timp real.

Real-Time Table

Timestamp per minute	Slot 1 Taken	Slot 2 Taken
02:45	true	true
02:45	false	true
02:40	true	true
02:40	true	false
02:15	true	true
02:15	true	false
02:15	false	true
02:09	true	true
02:09	true	false
02:00	false	false

- Am adaugat si un heatmap, care imi afiseaza numarul de masuratori in perioada selectata



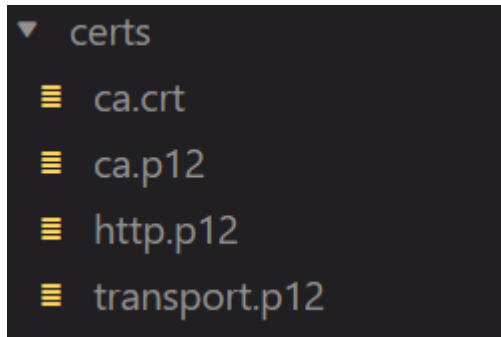
5. Securitate

Criptare

Am configurat **SSL/TLS** pentru ElasticSearch:

- Am generat fisierele http.p12 si transport.p12 in container-ul care ruleaza ElasticSearch si le-am configurat ca variabile de mediu in docker-compose.
- Am generat un certificat(ca.crt) si l-am exportat pentru a-l utiliza in container-ul care ruleaza Kibana si in sever-ul de Python pentru a facilita conexiunea HTTPS.

Lista de certificate utilizate in cadrul aplicatiei:



Configurarile de Securitate sunt prezente in fisierul docker-compose.yml

```
services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.10.1
    container_name: elasticsearch
    environment:
      - discovery.type=single-node
      - xpack.security.enabled=true
      - xpack.security.http.ssl.enabled=true
      - xpack.security.http.ssl.keystore.path=/usr/share/elasticsearch/config/certs/http.p12
      - xpack.security.http.ssl.truststore.path=/usr/share/elasticsearch/config/certs/http.p12
      - xpack.security.transport.ssl.enabled=true
      - xpack.security.transport.ssl.verification_mode=certificate
      - xpack.security.transport.ssl.keystore.path=/usr/share/elasticsearch/config/certs/transport.p12
      - xpack.security.transport.ssl.truststore.path=/usr/share/elasticsearch/config/certs/transport.p12
      - ELASTIC_PASSWORD=changeme
      - TZ=Europe/Bucharest
    ulimits:
      memlock:
        soft: -1
        hard: -1
    ports:
      - "9200:9200"
    volumes:
      - es_data:/usr/share/elasticsearch/data
      - ./certs:/usr/share/elasticsearch/config/certs
    networks:
      - elastic

  kibana:
    image: docker.elastic.co/kibana/kibana:8.10.1
    container_name: kibana
    ports:
      - "5601:5601"
    environment:
      - ELASTICSEARCH_HOSTS=https://elasticsearch:9200
      - XPACK_ENCRYPTEDSAVEDOBJECTS_ENCRYPTIONKEY=JVwm81K0evKt3EwDutbTDbn1pxjUSWJPFoFTorUGXx8=
      - ELASTICSEARCH_USERNAME=kibana_system
      - ELASTICSEARCH_PASSWORD=password
      - ELASTICSEARCH_SSL_CERTIFICATEAUTHORITIES=/usr/share/kibana/config/certs/ca.crt
      - PACK_SECURITY_ENABLED=true
    depends_on:
      - elasticsearch
    volumes:
      - ./certs:/usr/share/kibana/config/certs
    networks:
      - elastic
```

Autentificare

ElasticSearch:

- Am activat autentificarea prin utilizatori (elastic, kibana_system) si parole securizate.

6. Provocari si solutii

O prima provocare a fost ca nu am mai lucrat cu Docker pana acum, si am considerat ca acest proiect este o oportunitate buna de a invata o noua tehnologie.

Initial nu stiam cum functioneaza containerele, volumele sau yml-urile. Asa ca, volumul pe care am salvat dashboard-urile, l-am create ca fiind Docker managed, fiind mai greu apoi de transferat pe alta statie. Cu greu am reusit sa gasesc niste comenzi care imi arhiveaza volumul din memoria Docker-ului, spre a putea fi apoi dezarhivat si adaugat inapoi pe statia noua pe care vreau sa rulez.

O alta provocare a fost defectarea a doi senzori de proximitate, fiind nevoit sa imi ajustez toate dashboard-urile.

De altfel, servomotorul pe care il folosesc trebuie alimentat la 5V, port pe care placuta mea nu il expune, dar, cu ajutorul internetului am gasit ca e posibil totusi sa poata trage suficienta putere de pe portul VBUS, lucru care a functionat.

Alte provocari au constat in utilizarea si securizarea Kibana, trebuind sa ma familiarizez cu sintaxa KQL (Kibana Query Language), dar si cu generarea certificatelor pentru SSL si TLS.

Instructiuni de rulare

IMPORTANT: Pentru restaurarea volumului de Docker care continue modificarile create si dashboard-urile, trebuie rulate urmatoarele comenzi pe Windows:

```
docker volume create parking-monitoring_es_data
```

```
docker run --rm -v es_data:/to_volume -v "${PWD}:/from_host" busybox sh -c "cd /to_volume && tar xvf /from_host/es_data_backup.tar"
```

1. In primul rand, trebuie sa pornim broker-ul de MQTT:

```
C:\Users\Carmen>mosquitto -v -c "C:\Program Files\mosquitto\mosquitto.conf"
1737204223: mosquitto version 2.0.20 starting
1737204223: Config loaded from C:\Program Files\mosquitto\mosquitto.conf.
1737204223: Opening ipv6 listen socket on port 1883.
1737204223: Opening ipv4 listen socket on port 1883.
1737204223: mosquitto version 2.0.20 running
```

2. Am configurat fisierul mosquitto.conf pentru a permite accesul neautentificat

```
⚙️ mosquitto.conf
1  allow_anonymous true
2  listener 1883
3  log_type all
4
```


3. Apoi, din directorul parking-monitoring, rulam

docker compose up -d:

```
PS C:\Users\Carmen\OneDrive\Documents\Facultate\anul 4\pr\parking-monitoring> docker compose up -d
time="2025-01-18T14:46:44+02:00" level=warning msg="C:\\Users\\Carmen\\OneDrive\\Documents\\Facultate
[+] Running 3/3
 ✓ Network parking-monitoring_elastic Created
 ✓ Container elasticsearch Started
 ✓ Container kibana Started
```

4. Astfel pornim containerele care ruleaza Kibana si Elasticsearch, urmand sa asteptam putin pana se initializeaza totul.

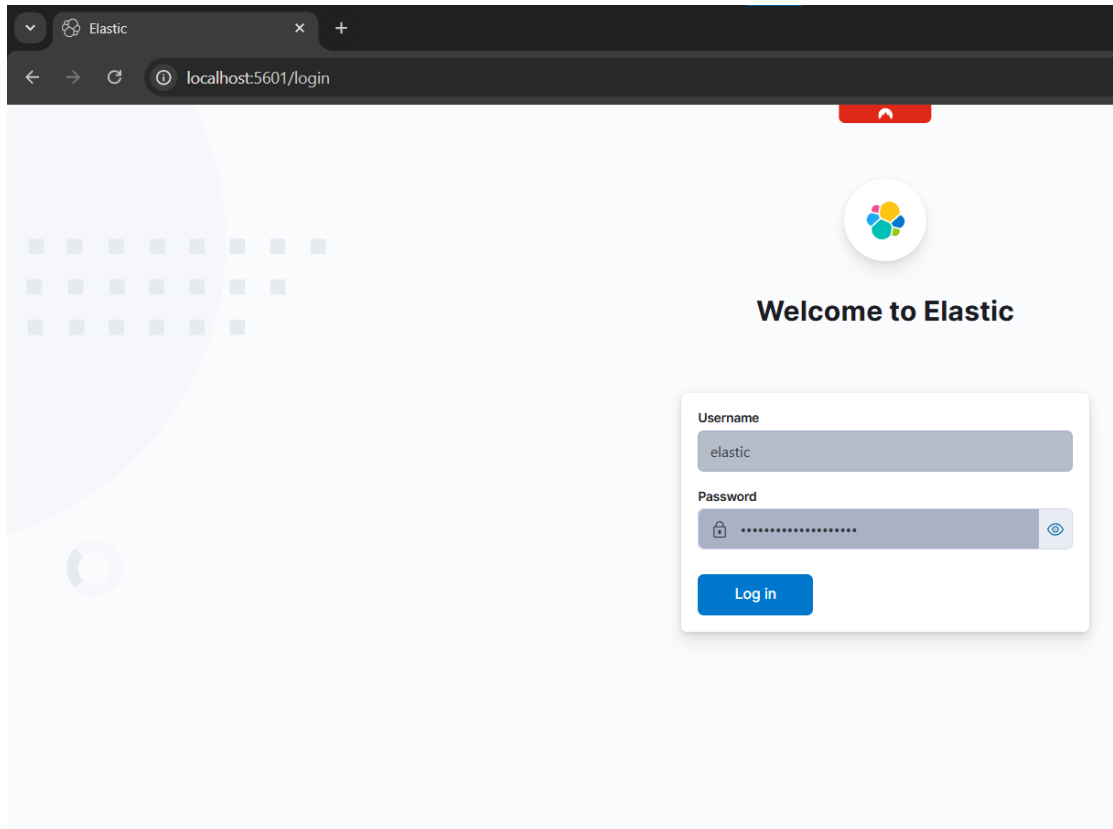
Putem verifica starea Kibana de exemplu, folosind comanda:

docker logs -f kibana

```
[2025-01-18T12:47:21.220+00:00][INFO ][status] Kibana is now available (was degraded)
[2025-01-18T12:47:21.241+00:00][INFO ][plugins.observability] SLO summary transforms already installed - skipping
[2025-01-18T12:47:21.251+00:00][INFO ][plugins.alerting] Installing index template .alerts-observability.logs.alerts-default-index-template
[2025-01-18T12:47:21.258+00:00][INFO ][plugins.alerting] Creating concrete write index - .internal.alerts-security.alerts-default-000001
[2025-01-18T12:47:21.274+00:00][INFO ][plugins.alerting] Creating concrete write index - .internal.alerts-observability.apm.alerts-default-000001
[2025-01-18T12:47:21.318+00:00][INFO ][plugins.observabilityAIAssistant.service] Creating concrete write index - .kibana-observability-ai-assistant-kb-000001
[2025-01-18T12:47:21.335+00:00][INFO ][plugins.alerting] Creating concrete write index - .internal.alerts-observability.metrics.alerts-default-000001
[2025-01-18T12:47:21.379+00:00][INFO ][plugins.observabilityAIAssistant.service] Successfully set up index assets
[2025-01-18T12:47:21.404+00:00][INFO ][plugins.alerting] Creating concrete write index - .internal.alerts-observability.logs.alerts-default-000001
[2025-01-18T12:47:21.955+00:00][INFO ][plugins.fleet] Fleet setup completed
[2025-01-18T12:47:21.956+00:00][INFO ][plugins.securitySolution] Dependent plugin setup complete - Starting ManifestTask
[2025-01-18T12:47:21.959+00:00][INFO ][plugins.securitySolution.endpoint.policyProtections] App feature [endpoint_policy_protections] is enabled. Nothing to do!
[2025-01-18T12:47:22.237+00:00][INFO ][plugins.synthetics] Installed synthetics index templates
```

Observam in partea de sus textul “[2025-01-18T12:47:21.220+00:00][INFO][status] Kibana is now available (was degraded)”.

Acest lucru inseamna ca ne putem conecta in browser pe localhost:5061



Aici ne vom loga cu user-ul **elastic** si parola din **enrolment-token.txt**.

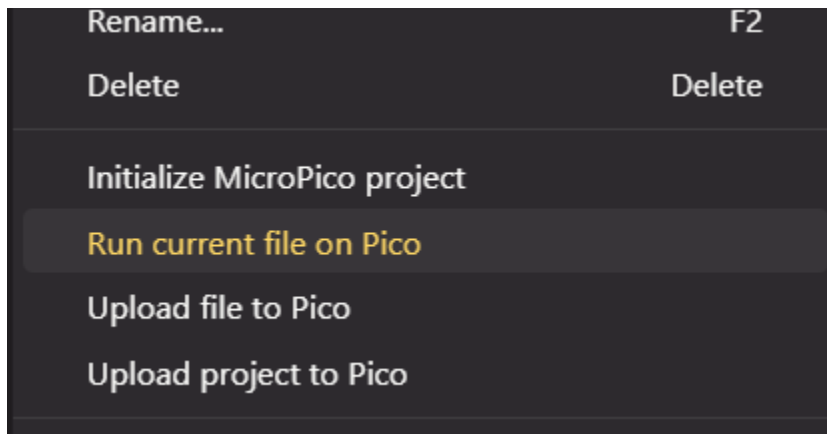
5. Pornim serverul de Python folosind comanda:
python es_proxy.py

```
PS C:\Users\Carmen\OneDrive\Documents\Facultate\anul 4\pr\parking-monitoring> python .\es_proxy.py
C:\Users\Carmen\OneDrive\Documents\Facultate\anul 4\pr\parking-monitoring\es_proxy.py:49: Deprecati
mqtt_client = Client()
```






6. Conectam placuta la statie, apoi rulam fisierul main-project.py pe Pico:

Deschidem un terminal de tip MicroPico vREPL in Visual Studio Code, dupa ce am instalat extensia MicroPico.

Click dreapta pe main-project, apoi *run current file on pico*



Structura repository Github <https://github.com/eric-zaharia/parking-monitoring>:

 eric-zaharia	added volume zip for restoring	95d5808 · 1 minute ago	 11 Commits
 certs	added alerts in kibana, created dashboards and added tls/ssl	last week	
 config	added alerts in kibana, created dashboards and added tls/ssl	last week	
 README.md	Update README.md	last month	
 docker-compose.yml	added alerts in kibana, created dashboards and added tls/ssl	last week	
 enrolment-token.txt	added alerts in kibana, created dashboards and added tls/ssl	last week	
 es_data_backup.tar	added volume zip for restoring	1 minute ago	
 es_proxy.py	added volume zip for restoring	1 minute ago	
 hcsr04.py	Add files via upload	last month	
 image.png	added alerts in kibana, created dashboards and added tls/ssl	last week	
 main-project.py	added manual controls on mqtt and updated pdf document...	last week	
 pr-proiect.pdf	added manual controls on mqtt and updated pdf document...	last week	
 simple.py	Add files via upload	last month	

Link YouTube: <https://www.youtube.com/watch?v=RIp9EpQVrdA>