Zaharia Eric-Alexandru, 345C1

Sistem de monitorizare parcare

1. Introducere

1. Descriere generala

Acest proiect implementează un sistem inteligent de monitorizare a parcării 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.

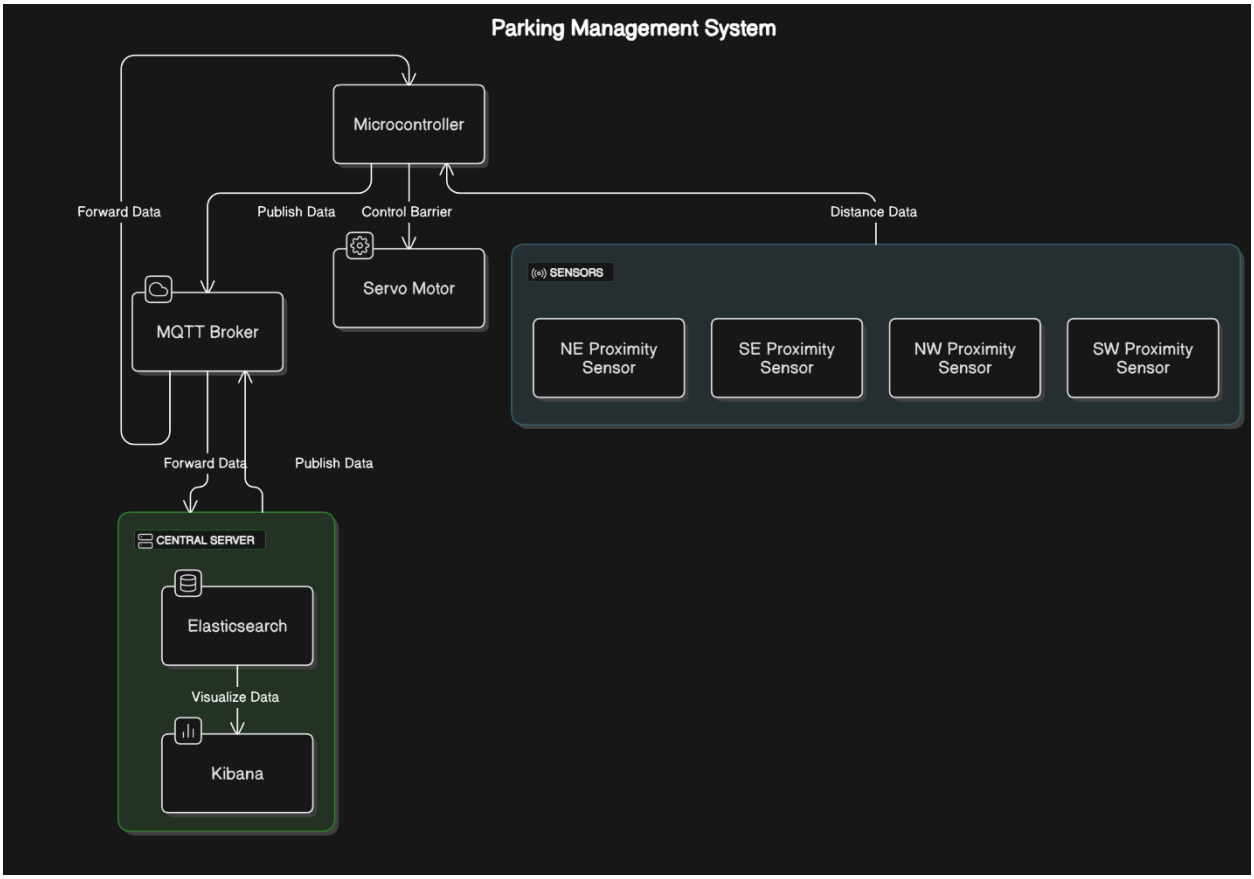
2. Actuator (servomotor):

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

3. Aplicație de control:

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

4. **Broker MQTT:**
    - Ruleaza 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.



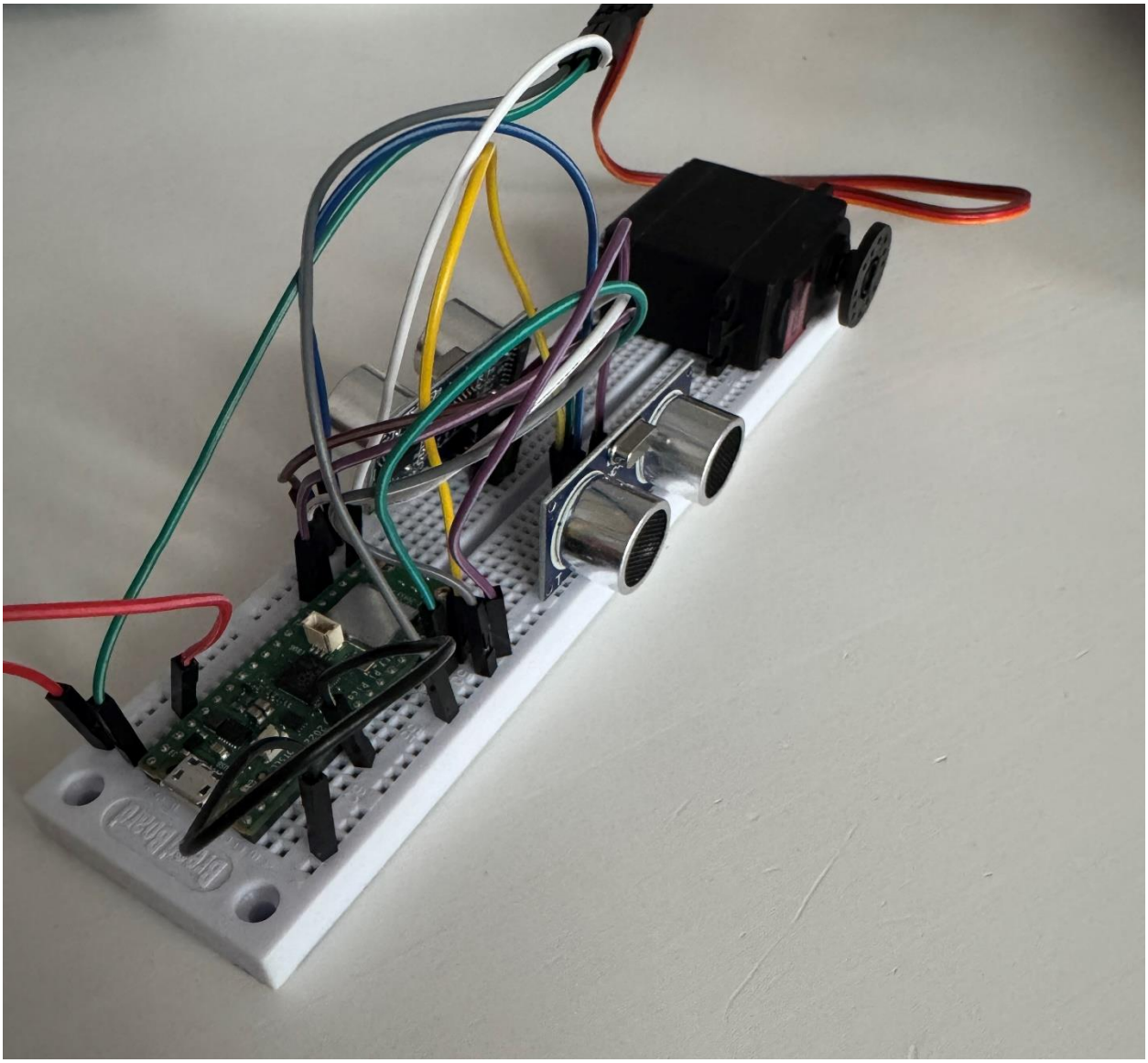
3. Implementare

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.

```python
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)
```

```python
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")
```

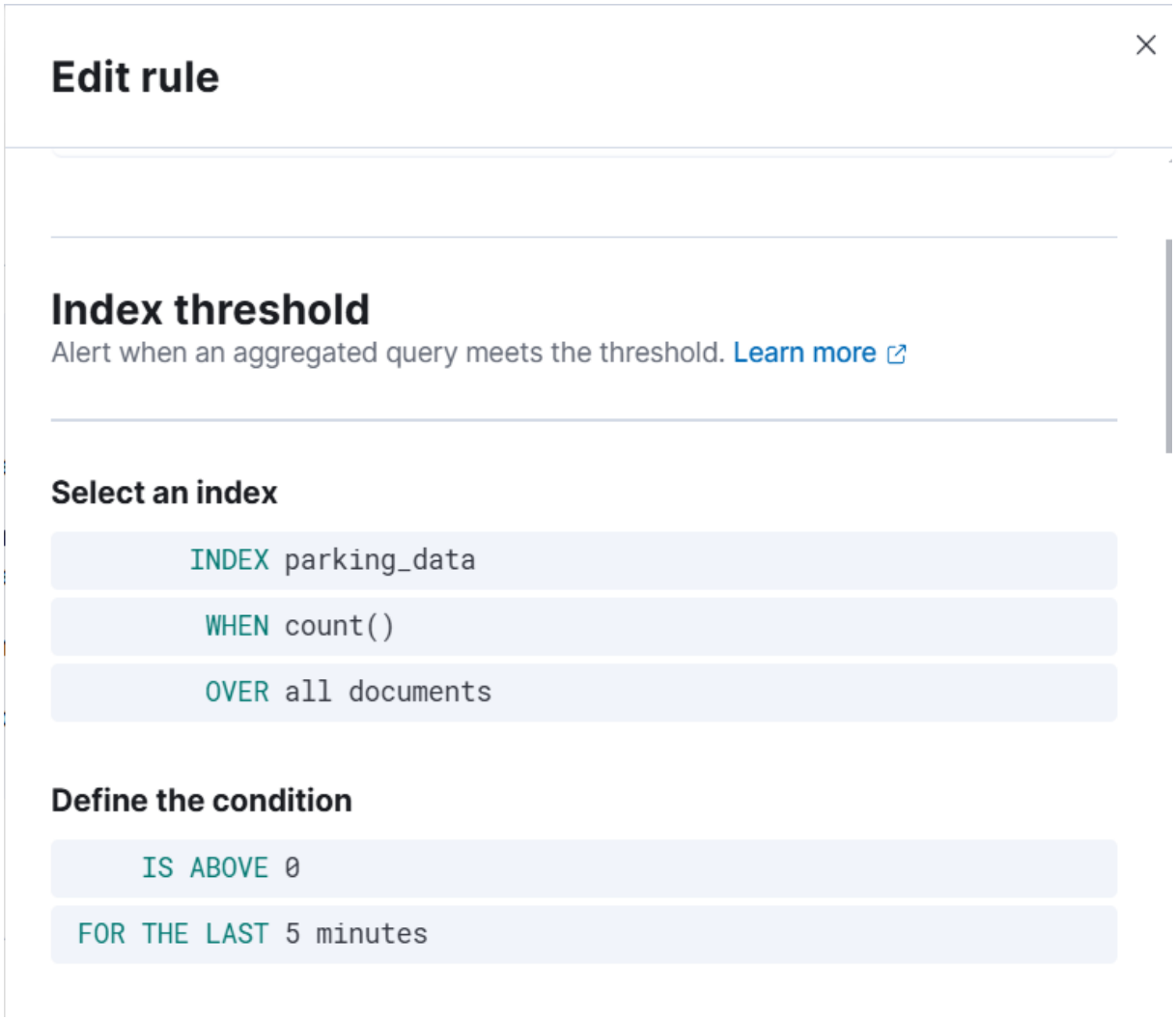
```
def on_message(client, userdata, msg):
    occupied_slots = {}
    sensors = msg.payload.decode().split(",")

    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")
```

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:



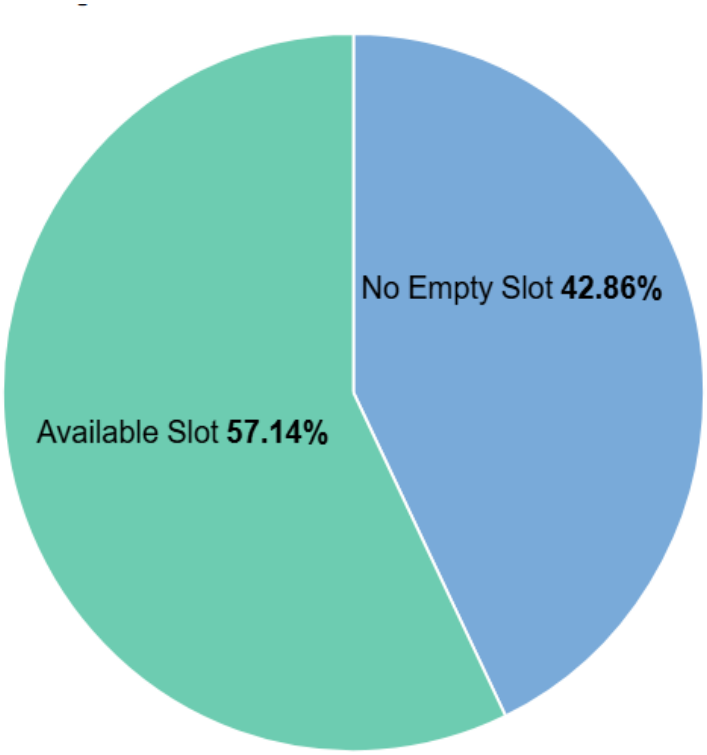
4. Vizualizare si Procesare de Date

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

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.

Autentificare

ElasticSearch:

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