Protocolos de internet

Trabajo Practico Nº4 Parte A-B

MQTT

Alumnos: Basara, Pejinakis, Soler G., Soler M., Tobio.

Consigna:

Instalar MQTT en Windows o Linux.

Usaremos Mosquitto con Broker MQTT

Instalar En Linux (Ubuntu)

sudo apt-get install mosquitto mosquitto-clients -y

Configurar El archivo de configurar se encuentra en: /etc/mosquitto/mosquitto.conf

Editarlo

Mosquitto como servidor escucha en el puerto 1883. Se puede cambiar en caso de ser necesario.

Arrancar y verificar el Servidor Mosquitto

sudo systemctl start mosquitto

sudo systemctl enable mosquitto

sudo systemctl status mosquitto

Suscribirse a un tema o tópico desde la terminal 1.

El cliente se suscribe al tema tempratura en el sitio1 y queda a la espera de recibir mensajes publicados en ese tema.

mosquitto_sub -h localhost -t sitio1/temperatura

Publicar datos desde la terminal 2

mosquitto_pub -h localhost -t sitio1/temperatura -m "Sitio1 Temp. = 22 C"

mosquitto_pub -h localhost -t sitio1/temperatura -m "Sitio1 Temp. = 23 C"

El broker (Mosquitto) escucha en el puerto 1883 para conexiones MQTT sin encriptación. Gestiona las suscripciones y publica mensajes a los clientes suscritos a los temas o tópicos específicos.

El cliente suscriptor (sería un servidor en este caso) se conecta al broker y se suscribe a uno o varios temas o tópicos. Queda en modo escuchs de mensajes a que el broker envíe a esos temas o tópicos.

El cliente publicador se conecta al broker y publica mensajes en uno o varios temas tópicos.El broker recibe el mensaje y lo reenvía a todos los clientes suscritos a esos tópicos.

Para hacer todo lo anterior en Windows:

Descargar e instalar Mosquitto desde

](https://mosquitto.org/)

Ejecutar el instalador

Asegúrarse de seleccionar la opción para instalar el servicio Mosquitto para que se inicie automáticamente.

Configuración del Path

Agregar la ruta de Mosquitto a las variables de entorno del sistema:

C:\Program Files\mosquitto).

Ejecutar Mosquitto Broker

Abrir una ventana de terminal (símbolo del sistema) como administrador y ejecutar:

sh

net start mosquitto

Para detener el broker:

sh

net stop mosquitto

Abrir dos ventanas de terminal (símbolo del sistema).

Seguir los mismos pasos explicados para Linux.

PDI - TP 4 - B - IoT MQTT (Entrega 24/06/2024 o en Primera Fecha de Final de Julio)

Una vez que tengan funcionando lo anterior deberán simular la publicación de datos de sensores tomando lo que está ingresado en la base datos. Por practicidad y seguridad se puede hacer una aplicación cliente servidor entre los sensores y una base de datos, para que luego un publicador mqtt tome los datos desde ahí.

Luego un suscriptor toma los datos desde el publicador y los pasa a otra base datos local y propia del publicador para su análisis.

Esto permite que varios suscritores usen los mismos datos.

Realizar la modificación de las rutas RestAPI para mostrar los datos de los sensores suscriptos de esta forma.

Como base pueden usar los siguientes scripts:

- mqtt_pub_r1.py
- mqtt_sub_r1.py

En la clase, de ser posible tomaremos datos reales y serán cargado en la BD a las que debe conectarse el publicador.

TP4 – A

Funcionamiento de parte A.

```
yr3b@DESKTOP-QFA0U13:/mnt/c/Users/gonza$ sudo systemctl start mosquitto
yr3b@DESKTOP-QFA0U13:/mnt/c/Users/gonza$ sudo systemctl enable mosquitto
Synchronizing state of mosquitto.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable mosquitto
yr3b@DESKTOP-QFA0U13:/mnt/c/Users/gonza$ sudo systemctl status mosquitto
● mosquitto.service - Mosquitto MQTT Broker
     Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
     Active: active (running) since Sat 2024-07-13 22:39:22 -03; 4min 54s ago
       Docs: man:mosquitto.conf(5)
             man:mosquitto(8)
   Main PID: 669 (mosquitto)
      Tasks: 1 (limit: 4663)
     Memory: 1.4M
     CGroup: /system.slice/mosquitto.service
             └─669 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

Jul 13 22:39:22 DESKTOP-QFA0U13 systemd[1]: Starting Mosquitto MQTT Broker...
Jul 13 22:39:22 DESKTOP-QFA0U13 systemd[1]: Started Mosquitto MQTT Broker.
```

```
yr3b@DESKTOP-QFA0U13:/mnt/c/Users/gonza$ mosquito_sub -h localhost -t sitio1/temperatura
Command 'mosquito_sub' not found, did you mean:
  command 'mosquitto_sub' from snap mosquitto (2.0.18)
  command 'mosquitto_sub' from deb mosquitto-clients (2.0.11-1ubuntu1)
See 'snap info <snapname>' for additional versions.
yr3b@DESKTOP-QFA0U13:/mnt/c/Users/gonza$ mosquitto_sub -h localhost -t sitio1/temperatura
Sitio 1 Temp. = 22C
Sitio 1 Temp. = 32C
```

```
yr3b@DESKTOP-QFA0U13:/mnt/c/Users/gonza$ mosquitto_pub -h localhost -t sitio1/temperatura -m "Sitio 1 Temp. = 22C"
yr3b@DESKTOP-QFA0U13:/mnt/c/Users/gonza$ mosquitto_pub -h localhost -t sitio1/temperatura -m "Sitio 1 Temp. = 32C"
yr3b@DESKTOP-QFA0U13:/mnt/c/Users/gonza$
```

TP4 – B

Para esta parte del TP, modificamos el código del trabajo anterior. Incluimos en el código de app.py librerías de SQLAlchemy para configurar la conexión a la base de datos. Modificamos el index() para mostrar la tabla sobre las lecturas de los sensores. Lo integramos en MQTT, agregando la ruta que devuelve los datos en formato JSON.

Luego utilizamos de base los códigos mqtt_pub_r1.py y mqtt_sub_r1.py. Agregamos configuración de Flask, la función publish_data() para publicar en los clientes los datos leídos de los sensores de la base de datos y sumamos manejo de excepciones para el primero y, en el segundo, usamos paho.mqtt.cliente para conectar y subscribirse a

sensores/datos, manejamos la conexión con el bróker con on_connect() y recibimos y procesamos los mensajes con la función on_message().

app.py

```
from flask import Flask, render_template, jsonify, redirect, url_for,
request, flash
from flask_login import LoginManager, UserMixin, login_user,
login_required, logout_user, current_user
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import DataRequired
from flask_sqlalchemy import SQLAlchemy
import os
import sqlite3
from datetime import datetime


app = Flask(__name__)
app.config['SECRET_KEY'] = 'mysecretkey'
db_path = os.path.join(os.getcwd(), 'datos_sensores.db')
app.config['SQLALCHEMY_DATABASE_URI'] = f"sqlite:///{db_path}"
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
app.app_context().push()


# Flask-Login setup
login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'


# User class
class User(UserMixin, db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(100), unique=True, nullable=False)
    password = db.Column(db.String(100), nullable=False)


# Database setup
def create_tables():
    db.create_all()


# Create tables
create_tables()


# Login manager user loader
```

```python
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))


# Forms
class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Login')


@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).first()

        if user and user.password == form.password.data:
                    login_user(user)
                    flash('Logged in successfully.', 'success')
                    return redirect(url_for('index'))
        else:
                    flash('Invalid username or password.', 'danger')
    return render_template('login.html', form=form)


@app.route('/logout')
@login_required
def logout():
    logout_user()
    flash('Logged out successfully.', 'success')
    return redirect(url_for('login'))


@app.route('/')
@login_required
def index():
    records = LecturaSensores.query.all()
    return render_template('tabla_sensores_para_editar.html',
records=records)


@app.route('/datos')
@login_required
def datos():
    records = LecturaSensores.query.all()
    return jsonify([{
        'co2': record.co2,
```

```
        'temp': record.temp,
        'hum': record.hum,
        'fecha': record.fecha,
        'lugar': record.lugar,
        'altura': record.altura,
        'presion': record.presion,
        'presion_nm': record.presion_nm,
        'temp_ext': record.temp_ext
    } for record in records])


# Route to delete a record
@app.route('/delete/<int:id>', methods=['POST'])
@login_required
def delete_record(id):
    record = LecturaSensores.query.get(id)
    if record:
        db.session.delete(record)
        db.session.commit()
        flash('Record deleted successfully.', 'success')
    else:
        flash('Record not found.', 'danger')
    return redirect(url_for('index'))


if __name__ == '__main__':
    app.run(debug=True, host='127.0.0.1', port=5000)
```

mqtt_pub_r1.py

```
import os
import json
import time
import logging
import paho.mqtt.client as mqtt
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from app import app,db
from datetime import datetime

# Configuración del logging para depuración
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s
%(levelname)s: %(message)s')

# Configuración de la aplicación Flask y la base de datos
# app = Flask(__name__)
# db_path = os.path.join(os.getcwd(), 'datos_sensores.db')
# app.config['SQLALCHEMY_DATABASE_URI'] = f"sqlite:///{db_path}"
# app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

```python
# db = SQLAlchemy(app)
# app.app_context().push()

class LecturaSensores(db.Model):
    __tablename__ = 'lectura_sensores'
    id = db.Column(db.Integer, primary_key=True)
    co2 = db.Column(db.Numeric(precision=10, scale=2))
    temp = db.Column(db.Numeric(precision=10, scale=2))
    hum = db.Column(db.Numeric(precision=10, scale=2))
    fecha = db.Column(db.Text)
    lugar = db.Column(db.Text)
    altura = db.Column(db.Numeric(precision=8, scale=2))
    presion = db.Column(db.Numeric(precision=8, scale=2))
    presion_nm = db.Column(db.Numeric(precision=8, scale=2))
    temp_ext = db.Column(db.Numeric(precision=8, scale=2))

# Configuración de MQTT
MQTT_BROKER = "localhost"
MQTT_PORT = 1883
MQTT_TOPIC = "sensores/datos"

client = mqtt.Client()
client.connect(MQTT_BROKER, MQTT_PORT, 60)

def publish_data():
    while True:
        try:
            records = LecturaSensores.query.all()
            for record in records:
                data = {
                    'co2': float(record.co2),
                    'temp': float(record.temp),
                    'hum': float(record.hum),
                    'fecha': record.fecha,
                    'lugar': record.lugar,
                    'altura': float(record.altura),
                    'presion': float(record.presion),
                    'presion_nm': float(record.presion_nm),
                    'temp_ext': float(record.temp_ext)
                }
                client.publish(MQTT_TOPIC, json.dumps(data))
                logging.debug(f"Datos publicados: {data}")
                time.sleep(5)  # Publicar cada 5 segundos
        except Exception as e:
            logging.error(f"Error al publicar datos: {e}")
            time.sleep(5)

if __name__ == '__main__':
    logging.info("Iniciando publicador de MQTT")
```

```
    publish_data()
```

mqtt_sub_r1.py

```
import json
import logging
import paho.mqtt.client as mqtt
from app import app,db
from mqtt_pub_r1 import LecturaSensores

# Configuración del logging para depuración
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s
%(levelname)s: %(message)s')

MQTT_BROKER = "localhost"
MQTT_PORT = 1883
MQTT_TOPIC = "sensores/datos"

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        logging.info("Conectado al broker MQTT")
        client.subscribe(MQTT_TOPIC)
    else:
        logging.error(f"Conexión fallida con código de resultado: {rc}")



def on_message(client, userdata, msg):
    try:
        data = json.loads(msg.payload.decode())
        lectura = LecturaSensores(
            co2=data['co2'],
            temp=data['temp'],
            hum=data['hum'],
            fecha=data['fecha'],
            lugar=data['lugar'],
            altura=data['altura'],
            presion=data['presion'],
            presion_nm=data['presion_nm'],
            temp_ext=data['temp_ext']
        )
        db.session.add(lectura)
        db.session.commit()
        logging.debug(f"Datos recibidos y guardados en la base de datos:
{data}")
    except Exception as e:
        logging.error(f"Error al procesar mensaje MQTT: {e}")
```

```
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message


try:
    client.connect(MQTT_BROKER, MQTT_PORT, 60)
    logging.info("Conectando al broker MQTT")
    client.loop_forever()
except Exception as e:
    logging.error(f"Error al conectar al broker MQTT: {e}")
```

Por último, adjuntamos los logs de las funciones correspondientes a MQTT, a modo de demostración del éxito en el funcionamiento del programa.

yr3b@DESKTOP-QFA0U13:/mnt/c/Users/gonza/protocolos$ python3 mqtt_pub_r1.py

/mnt/c/Users/gonza/protocolos/mqtt_pub_r1.py:40: DeprecationWarning: Callback API version 1 is deprecated, update to latest version

client = mqtt.Client()

2024-07-13 23:57:58,536 INFO: Iniciando publicador de MQTT

2024-07-13 23:58:08,553 DEBUG: Datos publicados: {'co2': 886.24, 'temp': 31.26, 'hum': 42.95, 'fecha': '10-Jun-2024 (03:25:53.484397)', 'lugar': 'Berlin', 'altura': 50.0, 'presion': 2.0, 'presion_nm': 2.0, 'temp_ext': 22.3}

2024-07-13 23:58:13,558 DEBUG: Datos publicados: {'co2': 678.51, 'temp': 22.9, 'hum': 60.75, 'fecha': '10-Jun-2024 (00:27:35.357755)', 'lugar': 'Lanus', 'altura': 2.0, 'presion': 2.0, 'presion_nm': 2.0, 'temp_ext': 22.3}

2024-07-13 23:58:18,564 DEBUG: Datos publicados: {'co2': 321.05, 'temp': 32.03, 'hum': 52.02, 'fecha': '10-Jun-2024 (01:52:02.700855)', 'lugar': 'Berazategui', 'altura': 100.0, 'presion': 2.0, 'presion_nm': 2.0, 'temp_ext': 22.3}

yr3b@DESKTOP-QFA0U13:/mnt/c/Users/gonza/protocolos$ python3 mqtt_sub_r1.py

/mnt/c/Users/gonza/protocolos/mqtt_sub_r1.py:44: DeprecationWarning: Callback API version 1 is deprecated, update to latest version

client = mqtt.Client()

2024-07-13 23:58:12,501 INFO: Conectando al broker MQTT

2024-07-13 23:58:12,501 INFO: Conectado al broker MQTT

2024-07-13 23:58:13,576 DEBUG: Datos recibidos y guardados en la base de datos: {'co2': 678.51, 'temp': 22.9, 'hum': 60.75, 'fecha': '10-Jun-2024 (00:27:35.357755)', 'lugar': 'Lanus', 'altura': 2.0, 'presion': 2.0, 'presion_nm': 2.0, 'temp_ext': 22.3}

2024-07-13 23:58:18,577 DEBUG: Datos recibidos y guardados en la base de datos: {'co2': 321.05, 'temp': 32.03, 'hum': 52.02, 'fecha': '10-Jun-2024 (01:52:02.700855)', 'lugar': 'Berazategui', 'altura': 100.0, 'presion': 2.0, 'presion_nm': 2.0, 'temp_ext': 22.3}

2024-07-13 23:58:23,582 DEBUG: Datos recibidos y guardados en la base de datos: {'co2': 886.24, 'temp': 31.26, 'hum': 42.95, 'fecha': '10-Jun-2024 (03:25:53.484397)', 'lugar': 'Berlin', 'altura': 50.0, 'presion': 2.0, 'presion_nm': 2.0, 'temp_ext': 22.3}