Programme - Envois MQTT

```
"""
parc-stub · MQTT sensor simulator
Publishes sine-wave sensor data + controleur connect/disconnect over mTLS.
"""

import json
import logging
import math
import os
import random
import ssl
import threading
import time

import paho.mqtt.client as mqtt

logger = logging.getLogger("stub")

CONFIG_PATH = os.getenv("CONFIG_PATH", "/data/config.json")

# ?? Persistence ??????????????????????????????????????????????????????????????
def _save(s: dict):
    """Save interval, bad_rate and serres layout (not log, not running)."""
    try:
        os.makedirs(os.path.dirname(CONFIG_PATH), exist_ok=True)
        payload = {
            "interval":  s["interval"],
            "bad_rate":  s["bad_rate"],
            "serres": [
                {k: v for k, v in serre.items() if k != "connected"}
                for serre in s["serres"]
            ],
        }
        tmp = CONFIG_PATH + ".tmp"
        with open(tmp, "w") as f:
            json.dump(payload, f, indent=2)
        os.replace(tmp, CONFIG_PATH)
    except Exception as e:
        logger.warning(f"Config save failed: {e}")

def _load() -> dict | None:
    try:
        with open(CONFIG_PATH) as f:
            return json.load(f)
    except FileNotFoundError:
        return None
    except Exception as e:
        logger.warning(f"Config load failed: {e}")
        return None


SENSORS = {
    "humiditeAmbiante":    {"min": 0,   "max": 100, "center": 60, "amp": 20},
```

```
    "humiditeSol":         {"min": 0,   "max": 100, "center": 55, "amp": 15},
    "temperatureAmbiante": {"min": -40, "max": 80,  "center": 22, "amp": 5},
}

def _default_serre(numero, mac=None, ip=None):
    return {
        "numero": numero,
        "mac": mac or _random_mac(),
        "ip": ip or f"192.168.1.{100 + numero}",
        "disconnect_rate": 0.0,   # probability per cycle of random disconnect
        "connected": False,       # current simulated connection state
        "bacs": [{"numero": 1}],
    }

def _random_mac():
    parts = [random.randint(0x00, 0xFF) for _ in range(6)]
    return ":".join(f"{p:02X}" for p in parts)

def _build_state() -> dict:
    saved = _load()
    if saved:
        serres = saved.get("serres", [])
        for s in serres:
            s["connected"] = False
            s.setdefault("disconnect_rate", 0.0)
        return {
            "running":  False,
            "interval": saved.get("interval", 10),
            "bad_rate": saved.get("bad_rate", 0.05),
            "serres":   serres,
            "log":      [],
        }
    return {
        "running": False,
        "interval": 10,
        "bad_rate": 0.05,
        "serres": [
            _default_serre(1, "24:D7:EB:38:DC:38", "192.168.1.101"),
            _default_serre(2, "24:D7:EB:AA:BB:CC", "192.168.1.102"),
        ],
        "log": [],
    }

state = _build_state()
# Per-serre manual connect/disconnect commands queued by API
# { serre_numero: "connect" | "disconnect" }
_cmd_queue: dict = {}

state_lock = threading.Lock()
_stop_event = threading.Event()
_thread = None


# ?? Logging ????????????????????????????????????????????????????????????????
def _log(msg: str, kind: str = "inf"):
    logger.info(msg)
    with state_lock:
        state["log"].append({"t": time.strftime("%H:%M:%S"), "msg": msg, "kind": kind})
        if len(state["log"]) > 150:
            state["log"].pop(0)
```

```
# ?? Controleur MQTT helpers ????????????????????????????????????????????
def _publish_connect(client: mqtt.Client, serre: dict):
    mac = serre["mac"]
    topic = f"controleur/{mac}"
    payload = json.dumps({
        "topic":  topic,
        "ip":     serre["ip"],
        "status": True,
    })
    client.publish(topic, payload, retain=False)
    _log(f"[CONN] controleur/{mac}  ip={serre['ip']}", "conn")


def _publish_disconnect(client: mqtt.Client, serre: dict):
    mac = serre["mac"]
    lwt_topic = f"controleur/{mac}/disconnect"
    client.publish(lwt_topic, "", retain=True)
    _log(f"[DISC] controleur/{mac}/disconnect (LWT)", "disc")


# ?? Sine / bad payload ??????????????????????????????????????????????????
def _sine_value(sensor: str, t: float, phase: float = 0.0) -> float:
    cfg = SENSORS[sensor]
    val = cfg["center"] + cfg["amp"] * math.sin(2 * math.pi * t / 120 + phase)
    return round(val, 2)

def _bad_payload(_t):
    kind = random.choice(["out_of_range", "unknown_sensor", "malformed", "invalid_value"])
    if kind == "out_of_range":
        return json.dumps({"humiditeAmbiante": 150.0})
    if kind == "unknown_sensor":
        return json.dumps({"co2Level": 400.0})
    if kind == "malformed":
        return b"\xff\xfe not utf-8"
    return json.dumps({"temperatureAmbiante": "hot"})


# ?? MQTT client ?????????????????????????????????????????????????????????
def _build_client() -> mqtt.Client:
    host     = os.getenv("MQTT_HOST", "mosquitto")
    port     = int(os.getenv("MQTT_PORT", 8883))
    ca       = os.getenv("CA_CERT",     "/certs/ca.crt")
    certfile = os.getenv("CLIENT_CERT", "/certs/client.crt")
    keyfile  = os.getenv("CLIENT_KEY",  "/certs/client.key")

    client = mqtt.Client(client_id="parc-stub", protocol=mqtt.MQTTv5)
    client.tls_set(ca_certs=ca, certfile=certfile, keyfile=keyfile,
                   tls_version=ssl.PROTOCOL_TLS_CLIENT)
    client.on_connect    = lambda c, u, f, rc, p: _log(f"MQTT connecté (rc={rc})", "inf")
    client.on_disconnect = lambda c, u, rc, p:    _log(f"MQTT déconnecté (rc={rc})", "inf")
    _log(f"Connexion à {host}:{port} …")
    client.connect(host, port, keepalive=60)
    return client


# ?? Publisher loop ??????????????????????????????????????????????????????
def _publish_loop():
    try:
```

```
    client = _build_client()
except Exception as e:
    _log(f"Échec connexion MQTT : {e}", "err")
    return

client.loop_start()
t0 = time.time()
phases: dict = {}

while not _stop_event.is_set():
    with state_lock:
        interval = state["interval"]
        bad_rate = state["bad_rate"]
        serres   = [dict(s) for s in state["serres"]]
        cmds     = dict(_cmd_queue)
        _cmd_queue.clear()

    t = time.time() - t0

    for serre in serres:
        sn  = serre["numero"]
        mac = serre["mac"]

        # ?? handle manual commands ??????????????????????????????????????
        cmd = cmds.get(sn)
        if cmd == "connect" and not serre["connected"]:
            _publish_connect(client, serre)
            with state_lock:
                for s in state["serres"]:
                    if s["numero"] == sn:
                        s["connected"] = True
            serre["connected"] = True
        elif cmd == "disconnect" and serre["connected"]:
            _publish_disconnect(client, serre)
            with state_lock:
                for s in state["serres"]:
                    if s["numero"] == sn:
                        s["connected"] = False
            serre["connected"] = False

        # ?? random disconnect ????????????????????????????????????????????
        if serre["connected"] and serre["disconnect_rate"] > 0:
            if random.random() < serre["disconnect_rate"]:
                _publish_disconnect(client, serre)
                with state_lock:
                    for s in state["serres"]:
                        if s["numero"] == sn:
                            s["connected"] = False
                serre["connected"] = False

        # ?? sensor data — only when connected ???????????????????????????
        if not serre["connected"]:
            continue

        for bac in serre.get("bacs", []):
            bn  = bac["numero"]
            key = (sn, bn)
            if key not in phases:
                phases[key] = {s: random.uniform(0, 2 * math.pi) for s in SENSORS}
```

```
                topic = f"serre/{sn}/bac/{bn}"
                if random.random() < bad_rate:
                    payload = _bad_payload(t)
                    _log(f"[BAD]  {topic} ? {payload!r}", "bad")
                else:
                    data    = {s: _sine_value(s, t, phases[key][s]) for s in SENSORS}
                    payload = json.dumps(data)
                    _log(f"[OK]   {topic} ? {payload}", "ok")
                try:
                    client.publish(topic, payload)
                except Exception as e:
                    _log(f"Erreur publish : {e}", "err")

        _stop_event.wait(interval)

    client.loop_stop()
    client.disconnect()
    _log("Stub arrêté.")


# ?? Public API ??????????????????????????????????????????????????????????????
def start():
    global _thread
    if _thread and _thread.is_alive():
        return
    _stop_event.clear()
    _thread = threading.Thread(target=_publish_loop, daemon=True)
    _thread.start()
    with state_lock:
        state["running"] = True
        _save(state)
    _log("Stub démarré.")

def stop():
    _stop_event.set()
    with state_lock:
        state["running"] = False
    _log("Arrêt demandé.")

def connect_serre(numero: int):
    with state_lock:
        _cmd_queue[numero] = "connect"
    _log(f"[CMD] connexion serre {numero} demandée", "inf")

def disconnect_serre(numero: int):
    with state_lock:
        _cmd_queue[numero] = "disconnect"
    _log(f"[CMD] déconnexion serre {numero} demandée", "inf")

def save_config():
    """Called by Flask after any config/serres mutation."""
    with state_lock:
        _save(state)
```

Revision #1
Created 2026-05-21 14:07:39 UTC by Clément
Updated 2026-05-21 14:08:36 UTC by Clément