

# Clément

Documentation projet SIGACS de Clément HERVOUET

- [Gestion de versions](#)
- [Docker - BROKER MQTT](#)
  - [Docker compose](#)
  - [Mosquitto](#)
  - [Application d'écriture en BDD](#)
  - [Programme "pont" complet](#)
- [Docker - Serveur WEB](#)
  - [Docker compose](#)
  - [Caddy](#)
  - [MySQL](#)
  - [Certificats](#)
- [Bouchon de test](#)
  - [Fonctionnement](#)
  - [Programme - API Bouchon](#)
  - [Programme - Envois MQTT](#)
  - [Docker compose](#)
- [Test interface Selenium](#)
  - [Test automatisés](#)
  - [Programme Python complet](#)
- [Procédure de démo](#)
  - [Script Selenium](#)

# Gestion de versions

## Arbre de commit GitHub en date du 22/05

```
* 5f2825e      conflits résolu
|\
| * 0783a22    (origin/main, origin/HEAD) Update README.md with new file layout
| * 2ce6140    Merge pull request #45 from clement-hervouet/docker-changes
| |\
| | * 82f6e59  (origin/docker-changes, docker-changes) UPDATE : upload des fichiers du
Broker docker
| * | 8415538  Merge pull request #44 from clement-hervouet/web-dev
| |\ \
| * \ \ 3d31a05      Merge pull request #43 from clement-hervouet/web-dev
| |\ \ \
| * \ \ \ 7f6b715    (main) Merge pull request #42 from clement-hervouet/web-dev
| |\ \ \ \
* | \ \ \ \ 13328dd  Merge branch 'web-dev' of https://github.com/clement-hervouet/SIGACS
into web-dev
|\ \ \ \ \
| * | | | | 3b5b033  UPDATE : ajout de la requête de population de la table capteur
| * | | | | 362f50e  HOTFIX : type des référence à id_utilisateur
| * | | | | c00dd1e  FIX : Changement des clés étrangères
| | |_|_| / /
| | / | | |
* | | | | 45d7749    FIX : suppression de app.py
* | | | | a575d37    UPDATE : menu dynamique et une partie des données récolté
* | | | | d3c6356    UPDATE : ajout de colonne + correction
| / / / /
* | | | | bb09a88    UPDATE : Traduction des pages de connexions
* | | | | 62826c1    FIX : système de login
* | | | | b214735    HOTFIX : bloc.empty --> bloc.vide (erreur)
* | | | | e9f40b6    FIX : references id incorrects
* | | | | 2a83e4a    UPDATE : user et parc sont désormais unis (references et serveur)
* | | | | 3d27ba6    UPDATE : requête SQL mise à jour (+lisible + lignes)
| |_| / /
| / | |
* | | | 6dfad6e  UPDATE : fichier de structure SQL refactoré
* | | | b83d308  FIX : style
* | | | bf475d7  REFACTOR : mise en place des formulaires de créations (complet) et d'éditions
(à compléter)
* | | | 6d521d0  UPDATE : Ajout "nouveau bac"
* | | | 505a485  UPDATE : formulaires structures
* | | | d03da48  UPDATE : mise en forme des formulaires
* | | | 4aa5b71  UPDATE : forms
* | | | 6ea84b1  UPDATE : menu polis avec des animations (icônes et transitions)
* | | | 1f73c2f  UPDATE : script séparé du fichier .php
* | | | 4d0dd66  UPDATE : menus déroulants dynamique
* | | | bf7e4d6  UPDATE : navigation tree complété pour avoir les lignes de jonctions
* | | | 07b6c94  UPDATE : sidebar header
* | | | 030a27c  UPDATE : Maquette principale du site + fonctionnement des pages
* | | | 2d02e57  SAVE : avant restart serveur
* | | | b00dcc9  UPDATE : ajout d'icônes de plantes tests
* | | | d6da955  FIX : faute d'orthographe
```

```

* | | | 419d9f1 HOTFIX : chemin de redirection erroné
| | / /
| / | |
* | | 10df665 FIX : chemin d'accès des icônes
* | | bda55ca FIX : chemin d'accès des icônes
* | | 0d46095 HOTFIX : unclosed condition
| | | * ele3782 (origin/MQTT-broker) FEATURE publication des données sur la BDD sans
certificat
| | | * 13784b9 DOC example secrets configuration
| | | * fc49bbd FEATURE ajout des topics + esp-now
| | | * 39a9ae6 Merge branch 'MQTT-broker' of https://github.com/clement-
hervouet/SIGACS into MQTT-broker
| | | | \
| | | | * 3abedd4 DOC diagramme de séquence MQTT
| | | | * f22e9bb REFACTOR
| | | | * | 601da87 REFACTOR + FEATURE Ajout code M5stack
| | | | /
| | | | * 720797a Merge branch 'MQTT-broker' of https://github.com/clement-
hervouet/SIGACS into MQTT-broker
| | | | \
| | | | * 1ea199a DOC Diagramme de séquence MQTT
| | | | * | d954db9 FIX NTP + ajout certificats
| | | | /
| | | | * 1286287 FIX NTP heure d'été + précision capteur
| | | | * 81864bd (MQTT-broker) REFACTOR : changement complet du modèle de broker
| | | | /
| | | | * f746d84 FEATURE ntp + optimisation sur 2 programmes
| | | | * 102ff1d FIX divergences
| | | | * eec5eae Merge branch 'MQTT-broker' of https://github.com/clement-
hervouet/SIGACS into MQTT-broker
| | | | \
| | | | * f100b0e REFACTOR changement de capteur de temperature v1.2
| | | | * 5717767 FIX résolution des conflits
| | | | * 020bdab UPDATE : server info
| | | | * | 67716fd UPDATE : noms de colonnes changé et programme python adapté
| | | | /
| | | | * ab2020b Merge branch 'main' of https://github.com/clement-hervouet/SIGACS
into MQTT-broker
| | | | \
| | | | * \ 382dbcd Merge branch 'MQTT-broker' of https://github.com/clement-
hervouet/SIGACS into MQTT-broker
| | | | \ \
| | | | | * | 5d5707e REFACTOR mise à niveaux de platformio.ini
| | | | | * | 56762c4 REFACTOR amélioration du code
| | | | | * | | e7dc3ce FIX : docker-compose
| | | | | * | | b53557f UPDATE : programme d'acquisition des données MQTT
| | | | | | * 60e2d4e (refs/stash) WIP on main: 26ab5d0 Merge pull request #41 from
clement-hervouet/web-dev
| | | | | | | | |
| | | | | | | | |
| | | | | | * e90ac45 index on main: 26ab5d0 Merge pull request #41 from clement-
hervouet/web-dev
| | | | | | | | /
| | | | | | | | |
| * | | | | 26ab5d0 Merge pull request #41 from clement-hervouet/web-dev
| | \ \ \ \ \
| | / / / / /
| / | | | | |
* | | | | | 0e0f196 HOTFIX : uncommented

```

```

| * | | | | 806c582 Merge pull request #40 from clement-hervouet/web-dev
| | \ \ \ \ \
| | / / / / /
| / | | | |
* | | | | 074c120 UPDATE : Ajout de nom prénom lors de l'enregistrement d'un
utilisateur
* | | | | 4aa13ff HOTFIX : index.php ne redirigeais pas correctement
* | | | | 3d664bd FIX : redirections via header
* | | | | 64c7270 FIX : chemins des icones corrigé
| * | | | | 7bdd71f Merge pull request #39 from clement-hervouet/web-dev
| | \ \ \ \ \
| | / / / / /
| / | | | |
* | | | | c13e1ea UPDATE : base de données user, ajout de colonnes nom prenom
* | | | | 21b92c1 UPDATE : système de login
* | | | | c47a4fd UPDATE : modification de la colonne is_admin à role dans la table
users
* | | | | e56e324 REFACTOR : chagement de l'architecture de la racine site
* | | | | 3labfb2 UPDATE : Page principale du site : menu ruban
| * | | | | 1d7390a Merge pull request #38 from clement-hervouet/web-dev
| | \ \ \ \ \
| | / / / / /
| / | | | |
* | | | | 979b07a UPDATE : Boutons de connexions page d'accueil / tableau de bord
* | | | | e732007 UPDATE : structure basique de la page principale
* | | | | 9452cb4 FIX : chemin vers style.css dans les fichiers php de connexions
* | | | | 2d44e4a FIX : rectification du chemin d'accès au fichier de config
* | | | | 4d2a321 UPDATE : mouvement des fichier de connexions vers dossier connexions
* | | | | ff95026 HOTFIX : info de connexion
| * | | | | 352aaff Merge pull request #37 from clement-hervouet/web-dev
| | \ \ \ \ \
| | / / / / /
| / | | | |
* | | | | 7b3b809 HOTFIX : dossier et ficheri de config
| * | | | | 493c079 Merge pull request #36 from clement-hervouet/web-dev
| | \ \ \ \ \
| | / / / / /
| / | | | |
* | | | | a2d84e7 UPDATE : docker compose du server web (caddy, php, phpmyadmin et
mysql)
* | | | | 342eed7 UPDATE : ajout de la colonne is_admin
* | | | | f189151 CHORE : ajout de la base du système d'utilisateur (bdd + pages web)
* | | | | 91a07a3 UPDATE : noms de colonne en français + colonne arrose
| * | | | | 26eb5c2 Merge pull request #34 from clement-hervouet/infra
| | \ \ \ \ \
| | * | | | 2beb608 (origin/infra) Update : user ssh , interface vlan , regles de
securiter
| | * | | | 1d91f10 Create conf SW-SERRES-1
| * | | | | b5339de Delete serre/PlatformIO/Projects/SIGACS directory
| | | _ | _ | _ | /
| | / | | | |
| * | | | | f64b060 Merge pull request #32 from clement-hervouet/web-dev
| | \ \ \ \ \
| | / / / / /
| / | | | |
* | | | | 757a60f FEATURE : ajout de la table error
* | | | | 855a47a FIX : correction nom colonne incorrect
* | | | | 44fe431 UPDATE : structure BDD
* | | | | a4905b7 CONCEPTION - structure bdd

```

```
| |_/ / / /
|/| | | |
| * | | | 5e4fc3e Merge branch 'MQTT-broker' of https://github.com/clement-
hervouet/SIGACS
| | \ \ \ \
| | | | / /
| | | / | |
| | * | | bba83af FEATURE - publish app
| | / / /
|/| | |
| * | | 0e71494 REFACTOR structure PlatformIO + capteurs MQTT fonctionnels - 4 fichiers
main.cpp - platformio.ini avec 4 environnements
| | | /
| | / |
| * | 865ea8f FEATURE Ajout du dossier PlatformIO + affichage des données avec les capteurs
| | /
| * 54deb50 Merge pull request #4 from clement-hervouet/docker-changes
| | \
| | /
|/|
* | bb7e6a4 update: broker files
|/
* a486ae4 Merge pull request #2 from clement-hervouet/chore/creation-racine
| \
| * de205c4 chore: création racines
* | 26c22e3 (origin/alan) Merge pull request #1 from clement-hervouet/chore/creation-
racine
| \ |
| * 7c9dcd3 update: readme et file-tree
|/
* e3d9881 Initial commit
```

# Docker - BROKER MQTT

Documentation des conteneurs docker déployé pour le projet SIGACS pour le fonctionnement du BROKER MQTT

# Docker compose

## docker-compose.yml

```
version: "3.9"

# =====
# Parc - MQTT Bridge stack
# =====
#
# Services:
#   mosquitto - MQTT broker
#   bridge    - Python MQTT?MySQL bridge (this repo)
#
# The MySQL/MariaDB server is expected to be external (already running).
# All database credentials and the broker address are passed via environment
# variables so no secret is baked into any image.
#
# =====

services:

# -----
# Mosquitto MQTT broker
# -----
mosquitto:
  image: eclipse-mosquitto:2
  container_name: mosquitto
  restart: unless-stopped
  volumes:
    - ./mosquitto/config/mosquitto.conf:/mosquitto/config/mosquitto.conf:ro
    - ./mosquitto/server-certs:/mosquitto/certs:ro
    - mosquitto-log:/mosquitto/log
  ports:
    - "${MQTT_EXPOSE_PORT:-8883}:8883"
  networks:
    - parc-net
  healthcheck:
    test: ["CMD-SHELL", "mosquitto_sub -t '$$SYS/broker/uptime' -C 1 -W 2 || exit 1"]
    interval: 10s
    timeout: 5s
    retries: 5

# -----
# MQTT ? MySQL bridge
# -----
bridge:
  build:
    context: .
    dockerfile: Dockerfile
```

```
image: parc-mqtt-bridge:latest
container_name: parc-bridge
restart: unless-stopped
volumes:
- ./mosquitto/server-certs/ca.crt:/certs/ca.crt:ro
- ./mosquitto/client-certs/client.crt:/certs/client.crt:ro
- ./mosquitto/client-certs/client.key:/certs/client.key:ro
depends_on:
  mosquitto:
    condition: service_healthy
environment:
  # --- MQTT ---
  MQTT_HOST:          mosquitto          # service name inside Docker network
  MQTT_PORT:          "8883"
  MQTT_KEEPALIVE:     "60"
  MQTT_USER:          ${MQTT_USER:-}
  MQTT_PASSWORD:      ${MQTT_PASSWORD:-}
  MQTT_RETRY_DELAY:   "5"

  # --- Database ---
  DB_HOST:            ${DB_HOST}
  DB_PORT:            ${DB_PORT:-3306}
  DB_NAME:            ${DB_NAME:-parc}
  DB_USER:            ${DB_USER}
  DB_PASSWORD:        ${DB_PASSWORD}
  DB_RETRY_DELAY:     "5"

  # --- Logging ---
  LOG_LEVEL:          ${LOG_LEVEL:-INFO}
networks:
- parc-net
logging:
  driver: "json-file"
  options:
    max-size: "10m"
    max-file: "5"

# -----
# Shared network
# -----
networks:
  parc-net:
    name: parc-net

# -----
# Volumes
# -----
volumes:
  mosquitto-log:
```

# Mosquitto

## mosquitto.conf

```
# Mosquitto configuration

# Plain listener - Partie 1
# utilisé uniquement par le healthcheck docker
listener 1883
allow_anonymous true

# TLS listener - Partie 2
listener 8883
protocol mqtt
allow_anonymous true
cafile /mosquitto/certs/ca.crt
certfile /mosquitto/certs/server.crt
keyfile /mosquitto/certs/server.key
require_certificate true

# Logging - Partie 3
log_dest file /mosquitto/log/mosquitto.log
log_dest stdout
log_type all
log_timestamp true
log_timestamp_format %Y-%m-%dT%H:%M:%S
connection_messages true
```

Le fichier de configuration est séparé en 3 parties

Partie	Explication
Partie 1	Paramètres des connexions non-sécurisées
Partie 2	Paramètres des connexions sécurisées
Partie 3	Gestions des paramètres des logs MQTT

## Partie 1

Paramètre	configuration	Explication
<code>listener</code>	1883	défini le port d'écoute sur lequel la configuration suivante va s'appliquer
<code>allow_anonymous</code>	false	<i>Explication ci-dessous</i>

## Partie 2

Paramètre	configuration	Explication
<code>listener</code>	8883	défini le port d'écoute sur lequel la configuration suivante va s'appliquer
<code>require_certificate</code>	false	autorise <b>uniquement</b> les connexions sécurisé sur le port du <code>listener</code> concerné (ici 8883)
<code>protocol</code>	mqtt	Défini le(s) protocole(s) en attente sur ce port
<code>allow_anonymous</code>	true	<i>Explication ci-dessous</i>
<code>cafile</code>	/mosquitto/certs/ca.crt	Chemin d'accès du certificat d'autorité du serveur, utilisé pour vérifier les clients
<code>certfile</code>	/mosquitto/certs/server.crt	Chemin d'accès du certificat public du serveur. Ce que Mosquitto présente aux clients lors de la connexion TLS
<code>keyfile</code>	/mosquitto/certs/server.key	Chemin d'accès de la clé privée du serveur

## paramètre important Partie 1 & 2 :

`allow_anonymous` :

`true` = mTLS — le client et le serveur prouvent leur identité mutuellement

`false` = TLS simple — le serveur prouve son identité au client

`tls_version` :

`tlsv1.2` = compatible ESP32 et programme Python

## Partie 3

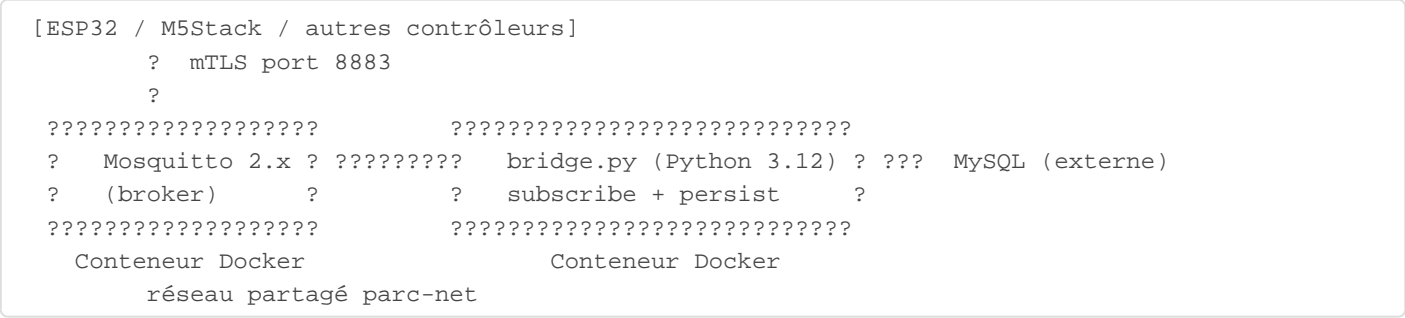
Paramètre	configuration	Explication
<code>log_dest</code>		Chemin de destination des logs

# Application d'écriture en BDD

## Parc – MQTT ? MySQL Bridge

Service Python qui fait le pont entre un broker MQTT Mosquitto et une base MySQL pour le système de monitoring de serres **SIGACS / Parc**.

### Architecture



Les deux conteneurs partagent le réseau Docker `parc-net`. Le serveur MySQL est externe (conteneurisé sur le web\_server).

### Stack

Composant	Technologie
Broker MQTT	Eclipse Mosquitto 2.x (Docker)
Bridge	Python 3.12 (Docker)
Base de données	MySQL
Bibliothèque MQTT	paho-mqtt 2.1.0
Bibliothèque DB	PyMySQL 1.1.1
TLS	mTLS — CA auto-signé, certificats par client
Conteneurisation	Docker Compose

# Topics MQTT

## Mesures capteurs

Pattern	Description
<code>serre/&lt;numero&gt;/bac/&lt;numero&gt;</code>	Mesures du bac <code>&lt;numero&gt;</code> dans la serre <code>&lt;numero&gt;</code>

`<numero>` correspond à la colonne `numero` dans les tables `serre` et `bac` (pas la clé primaire).

### Payload JSON :

```
{"humiditeAmbiante": 65.3, "temperatureAmbiante": 22.1, "humiditeSol": 45.0}
```

Plusieurs capteurs peuvent coexister dans un même payload.

### Capteurs supportés :

Clé JSON	<code>id_capteur</code>	Unité	Min	Max
<code>humiditeAmbiante</code>	1	%	0	100
<code>humiditeSol</code>	2	%	0	100
<code>temperatureAmbiante</code>	3	°C	-40	80

## Contrôleurs

Pattern	Direction	Déclencheur
<code>controleur/&lt;MAC&gt;</code>	Contrôleur → Broker	À chaque connexion du contrôleur
<code>controleur/&lt;MAC&gt;/disconnect</code>	Broker → Bridge	Déconnexion du contrôleur (LWT Mosquitto)

`<MAC>` est l'adresse MAC Wi-Fi du contrôleur au format `XX:XX:XX:XX:XX:XX`. Elle sert d'identifiant unique (`id_controleur`) en base de données.

### Payload de connexion (`controleur/<MAC>`) :

```
{"ip": "192.168.42.85", "status": true}
```

### Payload de déconnexion (`controleur/<MAC>/disconnect`) :

(payload vide)

Le bridge passe automatiquement `status = false` en base à la réception de ce message.

# Gestion des erreurs

Toutes les erreurs sont persistées dans la table `error` :

<code>type_erreur</code>	Déclencheur
<code>BAC_NOT_FOUND</code>	Numéro de serre ou bac introuvable en base
<code>UNKNOWN_SENSOR</code>	Clé JSON absente de la liste des capteurs connus
<code>INVALID_PAYLOAD</code>	Erreur de décodage JSON, payload non-objet, ou champ manquant
<code>INVALID_VALUE</code>	Valeur non numérique pour un capteur, ou type de champ invalide
<code>VALUE_OUT_OF_RANGE</code>	Valeur clampée au min/max du capteur
<code>INVALID_ENCODING</code>	Payload non UTF-8
<code>MQTT_DISCONNECT</code>	Déconnexion inattendue du broker
<code>DB_ERROR</code>	Erreur transitoire de requête base de données
<code>DB_INSERT_ERROR</code>	Échec d'insertion en base
<code>UNEXPECTED_ERROR</code>	Exception non gérée (attrape-tout)

Quand une valeur est hors limites : la valeur clampée (min ou max) est écrite dans `mesure` **et** une erreur est insérée.

## TLS / mTLS

### Architecture des certificats

```
ESP32 / M5Stack / autres clients
  ? mTLS
  ?
Mosquitto :8883 ??? server.crt (CN=mosquitto, SAN=DNS:mosquitto, IP:x.x.x.x)
  ?
  ? mTLS
  ?
bridge.py ????? présente client.crt (CN=mqtt-bridge)
```

### Fichiers de certificats

Fichier	Utilisé par	Monté à
<code>server-certs/ca.crt</code>	Mosquitto + bridge + clients ESP32	<code>/mosquitto/certs/ca.crt</code> et <code>/certs/ca.crt</code>
<code>server-certs/server.crt</code>	Mosquitto	<code>/mosquitto/certs/server.crt</code>
<code>server-certs/server.key</code>	Mosquitto	<code>/mosquitto/certs/server.key</code>
<code>client-certs/client.crt</code>	Bridge	<code>/certs/client.crt</code>
<code>client-certs/client.key</code>	Bridge	<code>/certs/client.key</code>

## Exigences critiques

- Le certificat serveur **doit** avoir `CN=mosquitto` **et** `subjectAltName=DNS:mosquitto,IP:<IP_LAN>` (Python SSL ignore le CN et ne vérifie que le SAN — les ESP32 connectent via IP LAN)
- Chaque nouveau client (ESP32, M5Stack, service...) obtient son propre certificat signé par le même CA
- Aucun changement de configuration Mosquitto nécessaire pour de nouveaux clients
- Taille des clés : 2048 bits (compatible ESP32 / M5Stack)
- Format des clés : PKCS#1 (`-----BEGIN RSA PRIVATE KEY-----`) → Sur OpenSSL 3.x, utiliser `openssl genrsa -traditional` pour forcer ce format

## `v3.ext` (requis pour la signature du certificat serveur)

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment
subjectAltName = @alt_names

[alt_names]
DNS.1 = mosquitto
IP.1 = 192.168.x.x
```

## Permissions

```
sudo chown 1883:1883 mosquitto/server-certs/server.key && chmod 600 mosquitto/server-certs/server.key
sudo chown 1001:1001 mosquitto/client-certs/client.key && chmod 600 mosquitto/client-certs/client.key
```

## Démarrage rapide

# 1. Prérequis

- Docker  $\geq$  24 avec le plugin Compose
- Serveur MySQL / MariaDB accessible avec le schéma `parc` appliqué
- Certificats générés (voir section TLS)

# 2. Configuration

```
cp .env.example .env
# Renseigner DB_HOST, DB_USER, DB_PASSWORD dans .env
```

# 3. Démarrage

```
docker compose up -d --build
```

# 4. Vérification

```
# Suivre les logs du bridge
docker compose logs -f bridge

# Tester une mesure capteur
docker run --rm --network parc-net eclipse-mosquitto:2 \
  mosquitto_pub -h mosquitto -p 8883 \
  -t "serre/1/bac/1" \
  -m '{"humiditeAmbiante": 65, "temperatureAmbiante": 22}'

# Tester une connexion contrôleur
docker run --rm --network parc-net eclipse-mosquitto:2 \
  mosquitto_pub -h mosquitto -p 8883 \
  -t "controleur/24:D7:EB:38:DC:38" \
  -m '{"ip": "192.168.42.85", "status": true}'
```

# Variables d'environnement

Variable	Défaut	Description
<code>DB_HOST</code>	<i>(requis)</i>	Hôte MySQL
<code>DB_PORT</code>	<code>3306</code>	Port MySQL
<code>DB_NAME</code>	<code>parc</code>	Nom de la base
<code>DB_USER</code>	<i>(requis)</i>	Utilisateur MySQL
<code>DB_PASSWORD</code>	<i>(requis)</i>	Mot de passe MySQL
<code>MQTT_HOST</code>	<code>mosquitto</code>	Nom d'hôte du broker

Variable	Défaut	Description
<code>MQTT_PORT</code>	<code>8883</code>	Port du broker (TLS)
<code>MQTT_KEEPALIVE</code>	<code>60</code>	Keepalive MQTT (s)
<code>MQTT_USER</code>	<i>(vide)</i>	Utilisateur broker (optionnel)
<code>MQTT_PASSWORD</code>	<i>(vide)</i>	Mot de passe broker (optionnel)
<code>DB_RETRY_DELAY</code>	<code>5</code>	Secondes entre tentatives DB
<code>MQTT_RETRY_DELAY</code>	<code>5</code>	Secondes entre tentatives MQTT
<code>LOG_LEVEL</code>	<code>INFO</code>	<code>DEBUG</code> / <code>INFO</code> / <code>WARNING</code> / <code>ERROR</code>

## Décisions de conception — bridge.py

- **Reconnexion DB** : `ensure_db()` appelé avant chaque requête, reconnecte transparentement en cas de coupure
- **Reconnexion MQTT** : `reconnect_delay_set(min=5, max=60)` + `loop_forever(retry_first_connection=True)`
- **Cache mémoire** : `_bac_cache` et `_capteur_cache` évitent les allers-retours DB sur chaque message ; invalidés à chaque reconnexion DB
- **Clampage de valeurs** : valeurs hors limites clampées au min/max, mesure insérée avec valeur clampée, erreur également insérée
- **Attrape-tout** : chaque callback `on_message` est enveloppé dans try/except pour qu'un seul message mal formé ne fasse jamais planter le bridge
- **Routage de topics** : `on_message()` route vers `process_controlleur_message()`, `set_controlleur_offline()` OU `process_message()` selon le topic reçu
- **LWT contrôleur** : le bridge écoute `controlleur/+/disconnect` — Mosquitto publie ce message automatiquement si un client se déconnecte brutalement

## Génération d'un nouveau certificat client

Pour chaque nouveau client (ESP32, M5Stack, service...) :

```
cd mosquitto

# Sur OpenSSL 3.x, utiliser -traditional pour forcer le format PKCS#1
openssl genrsa -traditional -out newclient.key 2048

openssl req -new -out newclient.csr -key newclient.key
# Common Name : nom descriptif du client (ex: m5-serre-1)
```

```
openssl x509 -req -in newclient.csr \  
-CA server-certs/ca.crt \  
-CAkey server-certs/ca.key \  
-CAcreateserial -out newclient.crt -days 720  
  
mkdir client-certs-newclient  
mv newclient.* client-certs-newclient/
```

Vérifier que le cert et la clé correspondent :

```
openssl x509 -noout -modulus -in newclient.crt | openssl md5  
openssl rsa -noout -modulus -in newclient.key | openssl md5  
# Les deux hashes doivent être identiques
```

## Commandes de debug fréquentes

```
# Suivre tous les logs  
docker compose logs -f  
  
# Logs d'un seul service  
docker compose logs -f bridge  
docker compose logs mosquito  
  
# Vérifier le SAN du certificat serveur  
openssl x509 -in mosquito/server-certs/server.crt -noout -ext subjectAltName  
  
# Vérifier la chaîne de confiance du certificat serveur  
openssl verify -CAfile mosquito/server-certs/ca.crt mosquito/server-certs/server.crt  
  
# Vérifier la connexion TLS depuis le serveur  
openssl s_client -connect 127.0.0.1:8883 -CAfile mosquito/server-certs/ca.crt  
  
# Vérifier correspondance cert / clé client  
openssl x509 -noout -modulus -in mosquito/client-certs/client.crt | openssl md5  
openssl rsa -noout -modulus -in mosquito/client-certs/client.key | openssl md5  
  
# Vérifier l'exposition des ports Docker  
docker ps | grep mosquito
```

## Structure des fichiers

```
mqt-bridge/  
??? bridge.py # Programme Python principal  
??? Dockerfile # Image du bridge  
??? docker-compose.yml # Définition du stack complet  
??? requirements.txt # Dépendances Python  
??? .env.example # Modèle de variables d'environnement  
??? mosquito/
```

```
??? config/
?   ??? mosquitto.conf           # Configuration du broker
??? server-certs/              # CA + certificats serveur
?   ??? ca.crt
?   ??? ca.key
?   ??? server.crt
?   ??? server.key
??? client-certs/              # Certificat client du bridge
?   ??? client.crt
?   ??? client.key
??? server-certs.sh            # Script de génération CA + serveur
??? clients-certs.sh           # Script de génération d'un cert client
??? v3.ext                      # Extension SAN pour la signature serveur
```

## Intégration ESP32 / M5Stack

- Se connecter à l'**IP LAN du serveur hôte** (pas au nom de service Docker `mosquitto`)
- Le port 8883 doit être exposé sur l'hôte dans `docker-compose.yml`
- Chaque appareil obtient son propre certificat client (même CA, nouveau key+CSR+crt, CN différent)
- Utiliser des clés 2048 bits (les clés 4096 bits peuvent dépasser la RAM de l'ESP32)
- Format clé privée : PKCS#1 (`-----BEGIN RSA PRIVATE KEY-----`) — générer avec `-traditional` sur OpenSSL 3.x
- Embarquer les certificats avec la syntaxe `R"EOF(... )EOF"` et `PROGMEM` pour stocker en flash
- Configurer le **Last Will Testament** sur `controleur/<MAC>/disconnect` avant `connect()`
- Publier les infos contrôleur sur `controleur/<MAC>` immédiatement après chaque connexion réussie

# Programme "pont" complet

```
"""
MQTT ? MySQL Bridge
=====
Subscribes to serre+/bac/+ and controleur/+ topics on a Mosquitto broker
and persists every measurement and controller info into the `parc` database.

Environment variables (injected via docker-compose):
MQTT_HOST          - hostname of the Mosquitto container (default: mosquitto)
MQTT_PORT          - broker port (default: 8883)
MQTT_KEEPALIVE    - keepalive in seconds (default: 60)
MQTT_USER         - optional broker username
MQTT_PASSWORD     - optional broker password

DB_HOST           - MySQL host (default: db)
DB_PORT           - MySQL port (default: 3306)
DB_NAME           - database name (default: parc)
DB_USER           - database user (default: parc)
DB_PASSWORD       - database password

DB_RETRY_DELAY    - seconds between DB reconnection attempts (default: 5)
MQTT_RETRY_DELAY  - seconds between MQTT reconnection attempts (default: 5)
LOG_LEVEL         - DEBUG / INFO / WARNING / ERROR (default: INFO)
"""

import json
import logging
import os
import re
import sys
import time
from datetime import datetime

import paho.mqtt.client as mqtt
import pymysql
import pymysql.cursors

# -----
# Logging setup
# -----
LOG_LEVEL = os.getenv("LOG_LEVEL", "INFO").upper()
logging.basicConfig(
    stream=sys.stdout,
    level=getattr(logging, LOG_LEVEL, logging.INFO),
    format="%(asctime)s %(levelname)-8s %(name)s %(message)s",
    datefmt="%Y-%m-%d %H:%M:%S",
)
logger = logging.getLogger("mqtt-bridge")

# -----
# Configuration from environment
# -----
```

```

MQTT_HOST      = os.getenv("MQTT_HOST",      "mosquitto")
MQTT_PORT      = int(os.getenv("MQTT_PORT",  "8883"))
MQTT_KEEPLIVE  = int(os.getenv("MQTT_KEEPLIVE", "60"))
MQTT_USER      = os.getenv("MQTT_USER",      None)
MQTT_PASSWORD  = os.getenv("MQTT_PASSWORD",  None)

DB_HOST        = os.getenv("DB_HOST",        "db")
DB_PORT        = int(os.getenv("DB_PORT",    "3306"))
DB_NAME        = os.getenv("DB_NAME",        "parc")
DB_USER        = os.getenv("DB_USER",        "parc")
DB_PASSWORD    = os.getenv("DB_PASSWORD",    "")

DB_RETRY_DELAY = int(os.getenv("DB_RETRY_DELAY", "5"))
MQTT_RETRY_DELAY = int(os.getenv("MQTT_RETRY_DELAY", "5"))

# Fixed capteur IDs as specified in the project brief
CAPTEUR_IDS: dict[str, int] = {
    "humiditeAmbiante": 1,
    "humiditeSol":      2,
    "temperatureAmbiante": 3,
}

# Topic patterns
TOPIC_RE       = re.compile(r"^serre/(\d+)/bac/(\d+)$")
CONTROLEUR_RE  = re.compile(r"^controleur/([^\d+]$")
DISCONNECT_RE  = re.compile(r"^controleur/([^\d]+)/disconnect$")

# -----
# Global DB connection
# -----
_db_conn: pymysql.connections.Connection | None = None

# -----
# DB helpers
# -----

def db_connect() -> pymysql.connections.Connection:
    """Connect to MySQL, retrying indefinitely on failure."""
    while True:
        try:
            conn = pymysql.connect(
                host=DB_HOST,
                port=DB_PORT,
                user=DB_USER,
                password=DB_PASSWORD,
                database=DB_NAME,
                charset="utf8mb4",
                autocommit=True,
                connect_timeout=10,
                cursorclass=pymysql.cursors.DictCursor,
            )
            logger.info("Connected to database %s@%s:%s/%s", DB_USER, DB_HOST, DB_PORT,
DB_NAME)
            return conn
        except pymysql.MySQLError as exc:
            logger.error("DB connection failed: %s - retrying in %ss", exc, DB_RETRY_DELAY)
            time.sleep(DB_RETRY_DELAY)

```

```

def ensure_db() -> pymysql.connections.Connection:
    """Return a live DB connection, reconnecting if necessary."""
    global _db_conn
    try:
        if _db_conn is None:
            raise pymysql.MySQLError("no connection yet")
        _db_conn.ping(reconnect=False)
    except pymysql.MySQLError:
        logger.warning("DB connection lost - reconnecting..")
        try:
            if _db_conn is not None:
                _db_conn.close()
        except Exception:
            pass
        _db_conn = db_connect()
        invalidate_caches()
    return _db_conn

def insert_error(type_erreur: str, message: str, valeur: str | None = None) -> None:
    """Write a row to the error table. Never raises - errors are only logged."""
    try:
        conn = ensure_db()
        with conn.cursor() as cur:
            cur.execute(
                "INSERT INTO error (type_erreur, message, valeur, erreur_a) "
                "VALUES (%s, %s, %s, %s)",
                (type_erreur[:64], message, valeur, datetime.now()),
            )
        logger.debug("Error logged ? [%s] %s", type_erreur, message)
    except Exception as exc:
        logger.critical("Could not write to error table: %s", exc)

def insert_mesure(bac_id: int, capteur_id: int, value: float) -> None:
    """Insert one measurement row."""
    conn = ensure_db()
    with conn.cursor() as cur:
        cur.execute(
            "INSERT INTO mesure (bac, value, mesure_a, capteur) VALUES (%s, %s, %s, %s)",
            (bac_id, value, datetime.now(), capteur_id),
        )

def upsert_controleur(mac: str, ip: str, status: bool) -> None:
    """
    Insert or update a controller row.
    The MAC address extracted from the topic is used as id_controleur (primary key).
    """
    conn = ensure_db()
    with conn.cursor() as cur:
        cur.execute(
            """
            INSERT INTO controleur (id_controleur, ip, status)
            VALUES (%s, %s, %s)
            ON DUPLICATE KEY UPDATE
                ip      = VALUES(ip),
                status = VALUES(status)
            """
        )

```

```

        (mac, ip, int(status)),
    )
    logger.info(
        "Contrôleur enregistré mac=%s ip=%s status=%s",
        mac, ip, status,
    )

def set_controleur_offline(mac: str) -> None:
    """Set a controller status to false (offline)."""
    try:
        conn = ensure_db()
        with conn.cursor() as cur:
            cur.execute(
                "UPDATE controleur SET status = 0 WHERE id_controleur = %s",
                (mac,)
            )
        logger.info("Contrôleur %s passé hors ligne", mac)
    except Exception as exc:
        msg = f"Erreur DB lors de la mise hors ligne du contrôleur {mac}: {exc}"
        logger.error(msg)
        insert_error("DB_ERROR", msg, None)

def resolve_bac(serre_numero: int, bac_numero: int) -> int | None:
    """
    Return id_bac for (serre.numero=serre_numero, bac.numero=bac_numero).
    Returns None when no matching row is found.
    """
    conn = ensure_db()
    with conn.cursor() as cur:
        cur.execute(
            """
            SELECT b.id_bac
            FROM   bac b
            JOIN   serre s ON s.id_serre = b.serre
            WHERE  s.numero = %s
                AND b.numero = %s
            LIMIT 1
            """,
            (serre_numero, bac_numero),
        )
        row = cur.fetchone()
        return row["id_bac"] if row else None

def get_capteur_limits(capteur_id: int) -> tuple[float | None, float | None]:
    """
    Return (valeurMinCapteur, valeurMaxCapteur) for a given capteur ID.
    Returns (None, None) when the capteur is not found.
    """
    conn = ensure_db()
    with conn.cursor() as cur:
        cur.execute(
            "SELECT valeurMinCapteur, valeurMaxCapteur FROM capteur WHERE id_capteur = %s",
            (capteur_id,)
        )
        row = cur.fetchone()
    if row:

```

```

        return row["valeurMinCapteur"], row["valeurMaxCapteur"]
    return None, None

# -----
# Cache
# -----
_bac_cache: dict[tuple[int, int], int | None] = {}
_capteur_cache: dict[int, tuple[float | None, float | None]] = {}

def cached_resolve_bac(serre_numero: int, bac_numero: int) -> int | None:
    key = (serre_numero, bac_numero)
    if key not in _bac_cache:
        _bac_cache[key] = resolve_bac(serre_numero, bac_numero)
    return _bac_cache[key]

def cached_capteur_limits(capteur_id: int) -> tuple[float | None, float | None]:
    if capteur_id not in _capteur_cache:
        _capteur_cache[capteur_id] = get_capteur_limits(capteur_id)
    return _capteur_cache[capteur_id]

def invalidate_caches() -> None:
    """Called after a DB reconnection so stale entries don't linger."""
    _bac_cache.clear()
    _capteur_cache.clear()
    logger.debug("Caches invalidated after DB reconnect")

# -----
# Message processing – mesures
# -----

def process_message(topic: str, raw_payload: str) -> None:
    """
    Full processing pipeline for one sensor MQTT message.

    1. Validate topic format
    2. Resolve serre / bac from DB
    3. Parse JSON payload
    4. For each measurement in the payload:
        a. Validate sensor name
        b. Validate value type
        c. Clamp value if out of bounds (and raise an error)
        d. Persist to mesure table
    """
    logger.debug("RECV topic=%s payload=%s", topic, raw_payload)

    # 1 - topic validation
    match = TOPIC_RE.match(topic)
    if not match:
        logger.warning("Ignored non-matching topic: %s", topic)
        return

    serre_numero = int(match.group(1))
    bac_numero = int(match.group(2))

    # 2 - bac resolution
    try:

```

```

        bac_id = cached_resolve_bac(serre_numero, bac_numero)
except Exception as exc:
    logger.error("DB error resolving bac for topic %s: %s", topic, exc)
    insert_error(
        "DB_ERROR",
        f"Impossible de résoudre bac pour le topic {topic}: {exc}",
        raw_payload,
    )
    return

if bac_id is None:
    msg = (
        f"Topic {topic} correspond à la serre n°{serre_numero} / bac n°{bac_numero} "
        f"qui n'existe pas dans la base de données."
    )
    logger.warning(msg)
    insert_error("BAC_NOT_FOUND", msg, raw_payload)
    return

# 3 - payload parsing
try:
    payload = json.loads(raw_payload)
    if not isinstance(payload, dict):
        raise ValueError("Le payload JSON n'est pas un objet")
except (json.JSONDecodeError, ValueError) as exc:
    msg = f"Payload JSON invalide sur le topic {topic}: {exc}"
    logger.warning(msg)
    insert_error("INVALID_PAYLOAD", msg, raw_payload)
    return

# 4 - iterate measurements
for sensor_name, raw_value in payload.items():
    _process_single_measure(topic, bac_id, sensor_name, raw_value, raw_payload)

def _process_single_measure(
    topic: str,
    bac_id: int,
    sensor_name: str,
    raw_value,
    raw_payload: str,
) -> None:
    """Process one sensor key/value pair from a payload."""

    # 4a - validate sensor name
    capteur_id = CAPTEUR_IDS.get(sensor_name)
    if capteur_id is None:
        msg = (
            f"Capteur inconnu « {sensor_name} » reçu sur le topic {topic}. "
            f"Capteurs attendus : {list(CAPTEUR_IDS.keys())}."
        )
        logger.warning(msg)
        insert_error("UNKNOWN_SENSOR", msg, raw_payload)
        return

    # 4b - validate value type
    try:
        value = float(raw_value)
    except (TypeError, ValueError):

```

```

    msg = f"Valeur non numérique pour {sensor_name} sur {topic}: {raw_value!r}"
    logger.warning(msg)
    insert_error("INVALID_VALUE", msg, str(raw_value))
    return

# 4c - clamp to capteur limits if needed
try:
    v_min, v_max = cached_capteur_limits(capteur_id)
except Exception as exc:
    logger.error("DB error fetching capteur limits for %s: %s", sensor_name, exc)
    insert_error(
        "DB_ERROR",
        f"Impossible de récupérer les limites du capteur {sensor_name}: {exc}",
        str(raw_value),
    )
    return

clamped_value = value

if v_min is not None and value < v_min:
    msg = (
        f"Valeur {value} pour {sensor_name} (id={capteur_id}) sur le topic {topic} "
        f"est inférieure au minimum autorisé ({v_min}). Valeur enregistrée : {v_min}."
    )
    logger.warning(msg)
    insert_error("VALUE_OUT_OF_RANGE", msg, str(value))
    clamped_value = v_min

elif v_max is not None and value > v_max:
    msg = (
        f"Valeur {value} pour {sensor_name} (id={capteur_id}) sur le topic {topic} "
        f"est supérieure au maximum autorisé ({v_max}). Valeur enregistrée : {v_max}."
    )
    logger.warning(msg)
    insert_error("VALUE_OUT_OF_RANGE", msg, str(value))
    clamped_value = v_max

# 4d - persist
try:
    insert_mesure(bac_id, capteur_id, clamped_value)
    logger.info(
        "Mesure enregistrée serre=%s bac=%s (id=%s) capteur=%s (id=%s) valeur=%s",
        topic.split("/")[1],
        topic.split("/")[3],
        bac_id,
        sensor_name,
        capteur_id,
        clamped_value,
    )
except Exception as exc:
    msg = f"Erreur DB lors de l'insertion de la mesure ({sensor_name}={value}) sur
{topic}: {exc}"
    logger.error(msg)
    insert_error("DB_INSERT_ERROR", msg, str(value))

# -----
# Message processing - contrôleurs
# -----

```

```

def process_controleur_message(topic: str, raw_payload: str) -> None:
    """
    Process a message received on controleur/<mac>.

    The MAC address from the topic IS the id_controleur (primary key).
    Expected payload: {"ip": "192.168.x.x", "status": true}
    """
    match = CONTROLEUR_RE.match(topic)
    if not match:
        return
    mac = match.group(1)

    # Parse JSON
    try:
        payload = json.loads(raw_payload)
        if not isinstance(payload, dict):
            raise ValueError("Le payload JSON n'est pas un objet")
    except (json.JSONDecodeError, ValueError) as exc:
        msg = f"Payload JSON invalide sur le topic {topic}: {exc}"
        logger.warning(msg)
        insert_error("INVALID_PAYLOAD", msg, raw_payload)
        return

    # Validate required fields
    missing = [k for k in ("ip", "status") if k not in payload]
    if missing:
        msg = f"Champs manquants dans le payload contrôleur {topic}: {missing}"
        logger.warning(msg)
        insert_error("INVALID_PAYLOAD", msg, raw_payload)
        return

    # Validate types
    try:
        ip = str(payload["ip"])
        status = bool(payload["status"])
    except (TypeError, ValueError) as exc:
        msg = f"Type de champ invalide dans le payload contrôleur {topic}: {exc}"
        logger.warning(msg)
        insert_error("INVALID_VALUE", msg, raw_payload)
        return

    # Persist
    try:
        upsert_controleur(mac, ip, status)
    except Exception as exc:
        msg = f"Erreur DB lors de l'enregistrement du contrôleur {mac}: {exc}"
        logger.error(msg)
        insert_error("DB_INSERT_ERROR", msg, raw_payload)

# -----
# MQTT callbacks
# -----

def on_connect(client: mqtt.Client, userdata, flags, reason_code, properties=None):
    if reason_code == 0:
        logger.info("MQTT connecté à %s:%s", MQTT_HOST, MQTT_PORT)
        client.subscribe("serre+/bac/", qos=1)
        client.subscribe("controleur/", qos=1)
        client.subscribe("controleur+/disconnect", qos=1)

```

```

        logger.info(
            "Abonné aux topics  serre/+/bac/+ | controleur/+ | controleur/+/disconnect"
        )
    else:
        logger.error("Échec de la connexion MQTT, code retour : %s", reason_code)

def on_disconnect(client: mqtt.Client, userdata, flags, reason_code, properties=None):
    if reason_code == 0:
        logger.info("Déconnexion MQTT propre")
    else:
        logger.warning(
            "Déconnexion MQTT inattendue (code=%s) - le client va retenter automatiquement",
            reason_code,
        )
        insert_error(
            "MQTT_DISCONNECT",
            f"Déconnexion MQTT inattendue (code={reason_code}). Reconnexion en cours...",
            None,
        )

def on_message(client: mqtt.Client, userdata, msg: mqtt.MQTTMessage):
    # Decode payload
    try:
        payload_str = msg.payload.decode("utf-8")
    except UnicodeDecodeError as exc:
        logger.warning("Payload non-UTF8 sur %s : %s", msg.topic, exc)
        insert_error(
            "INVALID_ENCODING",
            f"Payload non-UTF8 sur {msg.topic}: {exc}",
            repr(msg.payload),
        )
    return

    try:
        # Contrôleur déconnecté : controleur/<mac>/disconnect
        if DISCONNECT_RE.match(msg.topic):
            mac = msg.topic.split("/")[1]
            set_controleur_offline(mac)

        # Contrôleur connecté : controleur/<mac>
        elif CONTROLEUR_RE.match(msg.topic):
            process_controleur_message(msg.topic, payload_str)

        # Mesure capteur : serre/X/bac/Y
    else:
        process_message(msg.topic, payload_str)

    except Exception as exc:
        logger.exception(
            "Erreur inattendue lors du traitement du message %s: %s", msg.topic, exc
        )
    try:
        insert_error("UNEXPECTED_ERROR", str(exc), payload_str)
    except Exception:
        pass

```

```

def on_log(client, userdata, level, buf):
    if level == mqtt.MQTT_LOG_ERR:
        logger.debug("MQTT internal: %s", buf)

# -----
# Entry point
# -----

def main() -> None:
    logger.info("=== MQTT Bridge démarrage ===")
    logger.info(
        "Config MQTT : %s:%s | Config DB : %s@%s:%s/%s",
        MQTT_HOST, MQTT_PORT,
        DB_USER, DB_HOST, DB_PORT, DB_NAME,
    )

    # Initial DB connection - blocks until successful
    global _db_conn
    _db_conn = db_connect()

    # Pre-warm capteur cache
    for name, cid in CAPTEUR_IDS.items():
        limits = cached_capteur_limits(cid)
        logger.info("Capteur %-22s (id=%s) : min=%-6s max=%s", name, cid, limits[0],
limits[1])

    # MQTT client setup
    client = mqtt.Client(
        mqtt.CallbackAPIVersion.VERSION2,
        client_id="mqtt-bridge",
        clean_session=True,
    )
    client.on_connect = on_connect
    client.on_disconnect = on_disconnect
    client.on_message = on_message
    client.on_log = on_log

    if MQTT_USER:
        client.username_pw_set(MQTT_USER, MQTT_PASSWORD)

    # TLS setup
    client.tls_set(
        ca_certs="/certs/ca.crt",
        certfile="/certs/client.crt",
        keyfile="/certs/client.key",
    )

    client.reconnect_delay_set(min_delay=MQTT_RETRY_DELAY, max_delay=60)

    # Connect loop - keep trying until the broker is up
    while True:
        try:
            logger.info("Connexion au broker MQTT %s:%s...", MQTT_HOST, MQTT_PORT)
            client.connect(MQTT_HOST, MQTT_PORT, MQTT_KEEPALIVE)
            break
        except (OSError, ConnectionRefusedError) as exc:
            logger.error(
                "Broker MQTT injoignable : %s - retry dans %ss", exc, MQTT_RETRY_DELAY
            )

```

```
time.sleep(MQTT_RETRY_DELAY)
```

```
logger.info("Entrée dans la boucle MQTT principale")  
client.loop_forever(retry_first_connection=True)
```

```
if __name__ == "__main__":  
    main()
```

# Docker - Serveur WEB

Documentation des conteneurs docker déployé pour le projet SIGACS pour le fonctionnement du serveur WEB

# Docker compose

## docker-compose.yml

```
# =====  
# Parc - Stack du serveur WEB  
# =====  
#  
# Services:  
#   caddy      - serveur web  
#   db         - serveur de base de données  
#   php        - interpréteur php  
#   phpmyadmin - Interface web d'administration MySQL  
#  
# =====  
  
services:  
# -----  
# Caddy  
# -----  
caddy:  
  image: caddy:2.11  
  container_name: caddy  
  restart: unless-stopped  
  ports:  
    - "80:80"  
    - "443:443"  
    - "443:443/udp"  
  volumes:  
    - /etc/docker/volumes/caddy/conf:/etc/caddy  
    - /etc/docker/volumes/caddy/site:/srv  
    - caddy_data:/data  
    - caddy_config:/config  
  
  depends_on:  
    - php  
    - db  
  
# -----  
# Serveur MySQL  
# -----  
  
db:  
  image: mysql:latest  
  container_name: caddy-db  
  restart: unless-stopped  
  ports:  
    - "3306:3306"  
  environment:  
    - MYSQL_ROOT_PASSWORD=stjolorient
```

```
- MYSQL_DATABASE=user_base
- MYSQL_USER=caddy
- MYSQL_PASSWORD=caddy
volumes:
- /etc/docker/volumes/caddy/caddy-db:/var/lib/mysql
```

```
# -----
# Interpréteur PHP
# -----
```

```
php:
  container_name: php
  restart: always
  build: ./php
  volumes:
    - /etc/docker/volumes/caddy/site:/srv
```

```
# -----
# PHPMyAdmin
# -----
```

```
phpmyadmin:
  image: phpmyadmin
  restart: always
  ports:
    - 8080:80
  environment:
    - PMA_ARBITRARY=1
    - PMA_HOST=caddy-db
    - PMA_PORT=3306
    - APACHE_PORT=33060
  depends_on:
    - db
    - caddy
```

```
# -----
# Volumes
# -----
volumes:
  caddy_data:
  caddy_config:
```

Docker - Serveur WEB

# Caddy

Docker - Serveur WEB

# MySQL

Docker - Serveur WEB

# Certificats

# Bouchon de test

Documentation du bouchon de test mis en place afin d'envoyer de manière fiable et paramétrable.

# Fonctionnement

# Simulateur MQTT — Documentation

“ **parc-stub** est un simulateur de capteurs IoT pour le projet SIGACS.  
Il publie des données de serre sur le broker MQTT afin de tester le pipeline complet sans matériel réel.

## Aperçu de l'architecture

```
????????????????????????????????????????????????????????????????????
?  Navigateur (votre PC)                                                    ?
?  http://<serveur>:5001  ?  Web UI                                          ?
????????????????????????????????????????????????????????????????????
?  REST /api/*                                                                ?
????????????????????????????????????????????????????????????????????
?  Docker · parc-stub                                                         ?
?  ??????????????????????  contrôle  ?????????????????????????????????? ?
?  ?  Flask API  ?  ??????????????  ?  Thread publisher  ?  ?
?  ?  app.py      ?                ?  stub.py            ?  ?
?  ??????????????????????          ?????????????????????????????????? ?
?                                     ?                                     ?
?  /data/config.json  ???  persistance  ??????????                        ?
????????????????????????????????????????????????????????????????????
?                                     ? MQTT mTLS :8883                    ?
????????????????????????????????????????????????????????????????????
?  Docker · mosquitto                                                         ?
????????????????????????????????????????????????????????????????????
?                                     ?                                     ?
????????????????????????????????????????????????????????????????????
?  Docker · bridge.py  ?  MySQL (parc)                                       ?
????????????????????????????????????????????????????????????????????
```

## Données simulées — capteurs

Le stub publie trois capteurs en **onde sinusoïdale** (cycle de 2 minutes).  
Chaque bac possède un déphasage aléatoire — les courbes diffèrent entre bacs.

Capteur	Clé JSON	Centre	Amplitude	Unité
Humidité ambiante	<code>humiditeAmbiante</code>	60	±20	%
Humidité du sol	<code>humiditeSol</code>	55	±15	%
Température ambiante	<code>temperatureAmbiante</code>	22	±5	°C

Les données capteurs ne sont publiées **que si le contrôleur de la serre est connecté**.

## Topic et payload capteurs

```
serre/<numero_serre>/bac/<numero_bac>
```

```
{"humiditeAmbiante": 67.4, "humiditeSol": 52.1, "temperatureAmbiante": 24.3}
```

## Données simulées — contrôleurs (M5Stack)

Chaque serre est associée à un contrôleur identifié par son adresse MAC.

Événement	Topic	Sens
Connexion	<code>controleur/&lt;MAC&gt;</code>	Stub → Broker
Déconnexion (LWT)	<code>controleur/&lt;MAC&gt;/disconnect</code>	Stub → Broker (retain)

## Payload de connexion

```
{  
  "topic": "controleur/24:D7:EB:38:DC:38",  
  "ip": "192.168.1.101",  
  "status": true  
}
```

## Payload de déconnexion (LWT)

Payload vide `""`, topic retain `controleur/<MAC>/disconnect`.

# Injection d'erreurs

Le taux d'erreur capteurs (0-100 %) déclenche aléatoirement l'un de ces 4 types :

Type	Payload envoyé	Erreur attendue côté bridge
<code>out_of_range</code>	<code>{"humiditeAmbiante": 150.0}</code>	<code>VALUE_OUT_OF_RANGE</code>
<code>unknown_sensor</code>	<code>{"co2Level": 400.0}</code>	<code>UNKNOWN_SENSOR</code>
<code>malformed</code>	<code>\xff\xfe ...</code> (non-UTF-8)	<code>INVALID_ENCODING</code>
<code>invalid_value</code>	<code>{"temperatureAmbiante": "hot"}</code>	<code>INVALID_VALUE</code>

## Web UI

Accessible sur `http://<IP_serveur>:5001`

Section	Rôle
<b>Contrôle global</b>	Démarrer / arrêter le stub, régler l'intervalle de publication et le taux d'erreurs capteurs
<b>Serres &amp; Contrôleurs</b>	Ajouter / supprimer des serres ; configurer MAC, IP, taux de déconnexion aléatoire ; connecter / déconnecter manuellement
<b>Journal</b>	Affichage en direct — vert = OK, orange = erreur capteur, bleu = connexion, rouge = déconnexion

## Couleurs du journal

Couleur	Signification
<input type="checkbox"/> Vert	Message capteur nominal
<input type="checkbox"/> Orange	Message capteur invalide
<input type="checkbox"/> Bleu	Connexion contrôleur
<input type="checkbox"/> Rouge	Déconnexion contrôleur (LWT)

“ Les numéros de serre et bac **doivent correspondre** aux colonnes `numero` dans la base de données, sinon le bridge logge `BAC_NOT_FOUND`.

# Persistance de la configuration

La configuration (serres, bacs, MAC, IP, intervalle, taux d'erreur) est sauvegardée automatiquement dans `/data/config.json` à chaque modification.

- Survit aux **refreshs** de l'interface
- Survit aux **redémarrages** du conteneur Docker
- L'état `connected` est toujours remis à `false` au redémarrage

Le fichier est stocké dans le volume Docker nommé `parc-stub-data`.

## API REST

Méthode	Route	Description
GET	<code>/api/state</code>	État complet (running, config, serres, log)
POST	<code>/api/start</code>	Démarre la publication
POST	<code>/api/stop</code>	Arrête la publication
POST	<code>/api/config</code>	Modifie <code>interval</code> (s) et <code>bad_rate</code> (0-1)
POST	<code>/api/serres</code>	Remplace la liste complète des serres/bacs
POST	<code>/api/serre/&lt;n&gt;/connect</code>	Connecte manuellement le contrôleur de la serre n°
POST	<code>/api/serre/&lt;n&gt;/disconnect</code>	Déconnecte manuellement le contrôleur de la serre n°

## Exemples

```
# Changer l'intervalle et le taux d'erreur
curl -X POST http://<serveur>:5001/api/config \
  -H "Content-Type: application/json" \
  -d '{"interval": 5, "bad_rate": 0.1}'

# Connecter manuellement la serre 1
curl -X POST http://<serveur>:5001/api/serre/1/connect

# Déconnecter manuellement la serre 2
curl -X POST http://<serveur>:5001/api/serre/2/disconnect

# Remplacer la configuration des serres
curl -X POST http://<serveur>:5001/api/serres \
```

```
-H "Content-Type: application/json" \  
-d  
'[{"numero":1,"mac":"24:D7:EB:38:DC:38","ip":"192.168.1.101","disconnect_rate":0.05,"bacs":[{"  
numero":1},{ "numero":2}]}]'
```

# Certificats mTLS

Le stub utilise un certificat client dédié généré par `setup.sh` :

```
mosquitto/stub-certs/  
??? stub.crt    ? certificat client (CN=parc-stub, 720 jours)  
??? stub.key    ? clé privée RSA 2048 bits (chmod 600)
```

Signé par la même CA que le reste du projet — aucun changement de config Mosquitto nécessaire.

“ **Erreur fréquente** : `[Errno 21] Is a directory` — un des chemins de certificat dans le volume Docker pointe vers un dossier. Vérifier avec `docker inspect parc-stub | grep -A 20 Mounts`.

# Commandes utiles

```
# Logs en direct  
docker logs -f parc-stub  
  
# Redémarrer sans rebuild  
docker compose -f parc-stub/docker-compose.stub.yml restart  
  
# Reconstruire après modification des sources  
docker compose -f parc-stub/docker-compose.stub.yml up -d --build  
  
# Arrêter et supprimer le conteneur  
docker compose -f parc-stub/docker-compose.stub.yml down  
  
# Inspecter les volumes montés  
docker inspect parc-stub | grep -A 20 Mounts  
  
# Lire la config persistée  
docker exec parc-stub cat /data/config.json
```

# Fichiers du projet

```
parc/
??? setup.sh           ? configuration initiale (certificats + docker-compose)
??? mosquito/
?   ??? stub-certs/
?     ??? stub.crt     ? certificat client du stub
?     ??? stub.key     ? clé privée (non committée)
??? parc-stub/
    ??? Dockerfile
    ??? docker-compose.stub.yml ? service + volume parc-stub-data
    ??? requirements.txt
    ??? stub.py         ? logique de publication + persistance
    ??? app.py          ? API Flask
    ??? templates/
        ??? index.html ? interface Web
```

---

*Projet SIGACS — BTS CIEL · Saint Joseph LaSalle · Lorient*  
*HERVOUET Clément · BANCQUART Alan · LE GOUALEC Titouan*





```
return render_template("index.html")
```

```
if __name__ == "__main__":  
    app.run(host="0.0.0.0", port=5000, debug=False)
```













# Docker compose

```
services:
  parc-stub:
    build:
      context: .
    container_name: parc-stub
    restart: unless-stopped
    networks:
      - parc-net
    ports:
      - "5001:5000"          # Web UI ? http://localhost:5001
    volumes:
      - ../mosquitto/server-certs/ca.crt:/certs/ca.crt:ro
      - ../mosquitto/stub-certs/stub.crt:/certs/client.crt:ro
      - ../mosquitto/stub-certs/stub.key:/certs/client.key:ro
      - parc-stub-data:/data          # persists config.json across restarts
    environment:
      MQTT_HOST: mosquitto
      MQTT_PORT: "8883"
      CONFIG_PATH: /data/config.json
    logging:
      driver: json-file
      options:
        max-size: "5m"
        max-file: "3"

networks:
  parc-net:
    external: true          # already declared in your main compose

volumes:
  parc-stub-data:         # config.json persisted here
```

# Test interface Selenium

# Test automatisés

## SIGACS – Documentation du script de démo Selenium

Script de démonstration automatisée de l'interface web SIGACS.

Pilote Firefox en mode graphique, remplit les formulaires caractère par caractère, soumet les données en base, puis marque une pause opérateur après chaque enregistrement.

---

### Prérequis

Outil	Version minimale
Python	3.10+
Firefox	toute version récente
geckodriver	compatible avec Firefox installé
selenium	4.x

Installation rapide :

```
pip install selenium
# geckodriver doit être dans le PATH
```

---

### Lancer le script

```
python demo_sigacs.py
```

Firefox s'ouvre automatiquement, maximisé.

Chaque étape s'affiche dans le terminal.

Après chaque écriture en base, le script se fige jusqu'à ce que l'opérateur appuie sur **Entrée**.

---

# Configuration

Tous les réglages sont regroupés en haut du fichier dans le bloc `CONFIG`.

```
BASE_URL      = "http://192.168.42.131"    # adresse du serveur web

USERNAME      = "demo_user"                # compte existant en base
PASSWORD      = "demo1234"

CHAR_DELAY    = 0.07                       # secondes entre chaque frappe clavier
FIELD_PAUSE   = 1.0                        # pause après avoir rempli un champ
ACTION_PAUSE  = 1.5                        # pause après un clic
SECTION_PAUSE = 2.5                        # pause entre deux étapes majeures
PAGE_WAIT     = 8                          # délai max d'attente d'un élément (WebDriverWait)
```

Modifier un seul nombre change le rythme de toute la démo.

## Scénario couvert

Étape	Fonction	Censé échouer
1	<code>test_login</code>	oui
2	<code>test_admin_login</code>	non
1	<code>creation_demo_user</code>	non
1	<code>test_login</code>	non
2	<code>test_new_serre</code>	non
3	<code>test_new_bac</code>	non
4	<code>test_new_culture</code>	non
5	<code>test_logout</code>	non

## Pause opérateur

Après chaque soumission validée, le terminal affiche :

```
? New Serre record written - check the DB / UI
? Inspect the result, then press [Enter] to continue...
```

Le navigateur reste ouvert et interactif pendant cette pause.

Appuyer sur **Entrée** reprend le scénario.

# Gérer un test censé échouer

Utiliser la fonction utilitaire `attempt()` dans `run_demo()` au lieu d'appeler la fonction directement.

```
def attempt(fn, *args, expect_fail=False, **kwargs):
    try:
        fn(*args, **kwargs)
        if expect_fail:
            print(f"    ? {fn.__name__} était censé échouer mais a réussi")
    except Exception as e:
        if expect_fail:
            print(f"    ? {fn.__name__} a échoué comme prévu : {e}")
        else:
            raise # échec inattendu ? stoppe la démo
```

Exemple d'utilisation dans `run_demo()` :

```
def run_demo() -> None:
    driver = make_driver()
    try:
        attempt(test_login, driver, expect_fail=True) # doit échouer
        attempt(test_admin_login, driver)           # crée le compte
        attempt(test_login, driver)                 # doit réussir
        ...
```

`expect_fail=True` → l'exception est capturée et affichée comme un succès attendu, le scénario continue.

Sans le flag → toute exception remonte et stoppe la démo normalement.

## Fonctions utilitaires

Fonction	Rôle
<code>slow_type(element, text)</code>	Tape le texte caractère par caractère avec <code>CHAR_DELAY</code>
<code>pause(duration)</code>	Attend <code>duration</code> secondes (défaut : <code>ACTION_PAUSE</code> )
<code>wait_resume(label)</code>	Affiche le label et attend <code>[Entrée]</code>
<code>wait_for(driver, by, value)</code>	Attend qu'un élément soit présent dans le DOM
<code>wait_clickable(driver, by, value)</code>	Attend qu'un élément soit cliquable
<code>wait_invisible(driver, by, value)</code>	Attend qu'un élément disparaisse
<code>make_driver()</code>	Crée et maximise un driver Firefox headful
<code>attempt(fn, *args, expect_fail)</code>	Exécute un test en capturant les échecs attendus



# Programme Python complet

```
"""
SIGACS -- Selenium demo stub
=====
Covers:
  1. Login
  2. Dashboard -- New Serre ? validates (DB write)
  3. Dashboard -- New Bac ? validates (DB write)
  4. Dashboard -- New Culture ? validates (DB write)
  5. Logout

After each DB-write step the script freezes until you
press [Enter] in the terminal -- perfect for a live demo.

Tune every timing constant in the CONFIG block below.
"""

import time
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.firefox.options import Options
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

# -----
# CONFIG -- change these freely
# -----
BASE_URL      = "http://192.168.42.131"

# Demo credentials (existing account)
ADMIN_USERNAME = "demo_admin"
ADMIN_PASSWORD = "admindemo1234"

USERNAME      = "demo_user"
PASSWORD      = "demo1234"

# Delays (seconds)
CHAR_DELAY    = 0.07 # between each keystroke
FIELD_PAUSE   = 1.0  # after filling a field
ACTION_PAUSE  = 1.5  # after clicking
SECTION_PAUSE = 2.5  # between major steps
PAGE_WAIT     = 8    # max wait for elements

# -----
# HELPERS
# -----

def slow_type(element, text: str) -> None:
    """Type text character by character."""
    element.clear()
    for char in text:
        element.send_keys(char)
```

```

        time.sleep(CHAR_DELAY)

def pause(duration: float = ACTION_PAUSE) -> None:
    time.sleep(duration)

def wait_resume(label: str) -> None:
    """Freeze the demo until the operator presses Enter."""
    print(f"\n    ? {label}")
    print("    ? Inspect the result, then press [Enter] to continue...")
    input()

def wait_for(driver, by, value, timeout=PAGE_WAIT):
    return WebDriverWait(driver, timeout).until(
        EC.presence_of_element_located((by, value))
    )

def wait_clickable(driver, by, value, timeout=PAGE_WAIT):
    return WebDriverWait(driver, timeout).until(
        EC.element_to_be_clickable((by, value))
    )

def wait_invisible(driver, by, value, timeout=PAGE_WAIT):
    """Wait until an element disappears (e.g. modal closes)."""
    return WebDriverWait(driver, timeout).until(
        EC.invisibility_of_element_located((by, value))
    )

def expected_failure(fn):
    def wrapper(*args, **kwargs):
        try:
            fn(*args, **kwargs)
            print(f"    ? {fn.__name__} was expected to fail but passed")
        except Exception as e:
            print(f"    ? {fn.__name__} failed as expected: {e}")
    return wrapper

# -----
# DRIVER SETUP
# -----

def make_driver() -> webdriver.Firefox:
    opts = Options()
    driver = webdriver.Firefox(options=opts)
    driver.maximize_window()
    return driver

# -----
# TEST BLOCKS
# -----

def attempt(fn, *args, expect_fail=False, **kwargs):
    try:

```

```

    fn(*args, **kwargs)
    if expect_fail:
        print(f"    ? {fn.__name__} was expected to fail but passed")
except Exception as e:
    if expect_fail:
        print(f"    ? {fn.__name__} failed as expected: {e}")
    else:
        raise # échec réel ? plante le demo normalement

def test_admin_login(driver) -> None:
    print("\n--- [1/5] LOGIN ---")
    driver.get(f"{BASE_URL}/login.php")
    pause(SECTION_PAUSE)

    slow_type(wait_for(driver, By.ID, "username"), ADMIN_USERNAME)
    pause(FIELD_PAUSE)

    slow_type(wait_for(driver, By.ID, "password"), ADMIN_PASSWORD)
    pause(FIELD_PAUSE)

    wait_clickable(driver, By.CSS_SELECTOR, "input[type='submit']").click()
    pause(SECTION_PAUSE)

    wait_for(driver, By.CLASS_NAME, "navigation_tree")
    print("    ? Logged in -- dashboard loaded")
    pause(SECTION_PAUSE)

def creation_demo_user(driver) -> None:
    print("\n--- [2/5] creation user ---")

    wait_clickable(driver, By.ID, "add_user").click()
    pause(SECTION_PAUSE)

    driver.get(f"{BASE_URL}/connexions/register.php")
    pause(SECTION_PAUSE)

    slow_type(wait_for(driver, By.ID, "username"), USERNAME)
    pause(FIELD_PAUSE)

    slow_type(wait_for(driver, By.ID, "prenom"), "Utilisateur")
    pause(FIELD_PAUSE)

    slow_type(wait_for(driver, By.ID, "nom"), "De-la-Démo")
    pause(FIELD_PAUSE)

    slow_type(wait_for(driver, By.ID, "password"), PASSWORD)
    pause(FIELD_PAUSE)

    slow_type(wait_for(driver, By.ID, "confirm_password"), PASSWORD)
    pause(FIELD_PAUSE)

    wait_clickable(driver, By.CSS_SELECTOR, "input[type='submit']").click()
    pause(SECTION_PAUSE)

    wait_clickable(driver, By.ID, "log_out").click()
    pause(SECTION_PAUSE)

def test_login(driver) -> None:
    print("\n--- [1/5] LOGIN ---")

```

```

driver.get(f"{BASE_URL}/login.php")
pause(SECTION_PAUSE)

slow_type(wait_for(driver, By.ID, "username"), USERNAME)
pause(FIELD_PAUSE)

slow_type(wait_for(driver, By.ID, "password"), PASSWORD)
pause(FIELD_PAUSE)

wait_clickable(driver, By.CSS_SELECTOR, "input[type='submit']").click()
pause(SECTION_PAUSE)

wait_for(driver, By.CLASS_NAME, "navigation_tree")
print("    ? Logged in -- dashboard loaded")
pause(SECTION_PAUSE)

def test_new_serre(driver) -> None:
    print("\n--- [2/5] NEW SERRE ---")

    wait_clickable(driver, By.ID, "newSerreBtn").click()
    pause(ACTION_PAUSE)

    slow_type(
        wait_for(driver, By.CSS_SELECTOR, "#modalContainer input[name='nom']"),
        "Serre Demo"
    )
    pause(FIELD_PAUSE)

    slow_type(
        wait_for(driver, By.CSS_SELECTOR, "#modalContainer input[name='adresse']"),
        "42 rue de la Demo, Lorient"
    )
    pause(FIELD_PAUSE)

    slow_type(
        wait_for(driver, By.CSS_SELECTOR, "#modalContainer input[name='surface']"),
        "25.5"
    )
    pause(FIELD_PAUSE)

    slow_type(
        wait_for(driver, By.CSS_SELECTOR, "#modalContainer input[name='bac']"),
        "3"
    )
    pause(FIELD_PAUSE)

    # --- SUBMIT ---
    wait_clickable(driver, By.ID, "formSubmit").click()
    pause(SECTION_PAUSE)

    print("    ? Serre submitted")
    wait_resume("New Serre record written -- check the DB / UI")

def test_new_bac(driver) -> None:
    print("\n--- [3/5] NEW BAC ---")

    wait_clickable(driver, By.ID, "newBacBtn").click()

```

```

pause(ACTION_PAUSE)

slow_type(
    wait_for(driver, By.CSS_SELECTOR, "#modalContainer input[name='nom']"),
    "Bac Demo A"
)
pause(FIELD_PAUSE)

slow_type(
    wait_for(driver, By.CSS_SELECTOR, "#modalContainer input[name='blocX']"),
    "4"
)
pause(FIELD_PAUSE)

slow_type(
    wait_for(driver, By.CSS_SELECTOR, "#modalContainer input[name='blocY']"),
    "3"
)
pause(FIELD_PAUSE)

slow_type(
    wait_for(driver, By.CSS_SELECTOR, "#modalContainer input[name='numero']"),
    "1"
)
pause(FIELD_PAUSE)

# --- SUBMIT ---
wait_clickable(driver, By.ID, "formSubmit").click()
pause(SECTION_PAUSE)

print("    ? Bac submitted")
wait_resume("New Bac record written -- check the DB / UI")

def test_new_culture(driver) -> None:
    print("\n--- [4/5] NEW CULTURE ---")

    wait_clickable(driver, By.ID, "newCultureBtn").click()
    pause(ACTION_PAUSE)

    # Plant name (first 'nom' input)
    all_nom = driver.find_elements(By.CSS_SELECTOR, "#modalContainer input[name='nom']")
    slow_type(all_nom[0], "Tomate cerise")
    pause(FIELD_PAUSE)

    # Latin name (second 'nom' input)
    if len(all_nom) > 1:
        slow_type(all_nom[1], "Solanum lycopersicum")
        pause(FIELD_PAUSE)

    # Humidity & temperature numeric fields
    fields = {
        "humidite-ambiante (min)": ("humidite-ambiante", 0, "40"),
        "humidite-ambiante (max)": ("humidite-ambiante", 1, "85"),
        "humidite-ambiante (opt)": ("humidite-ambiante", 2, "65"),
        "humidite-sol (min)": ("humidite-sol", 0, "30"),
        "humidite-sol (max)": ("humidite-sol", 1, "80"),
        "humidite-sol (opt)": ("humidite-sol", 2, "60"),
        "temperature (min)": ("temperature-ambiante", 0, "15"),
    }

```

```

        "temperature (max)":      ("temperature-ambiante", 1, "32"),
        "temperature (opt)":     ("temperature-ambiante", 2, "22"),
    }

for label, (name, idx, val) in fields.items():
    inputs = driver.find_elements(
        By.CSS_SELECTOR, f"#modalContainer input[name='{name}']"
    )
    if idx < len(inputs):
        slow_type(inputs[idx], val)
        pause(FIELD_PAUSE * 0.5)

# Temps de pousse
temps = driver.find_elements(By.CSS_SELECTOR, "#modalContainer input[name='temps']")
if temps:
    slow_type(temps[0], "90")
    pause(FIELD_PAUSE)

# --- SUBMIT ---
wait_clickable(driver, By.ID, "formSubmit").click()
pause(SECTION_PAUSE)

print("    ? Culture submitted")
wait_resume("New Culture record written -- check the DB / UI")

def test_logout(driver) -> None:
    print("\n--- [5/5] LOGOUT ---")

    pause(ACTION_PAUSE)
    wait_clickable(driver, By.CSS_SELECTOR, "a[href='logout.php']").click()
    pause(SECTION_PAUSE)

    wait_for(driver, By.ID, "username")
    print("    ? Logged out -- back on login page")
    pause(SECTION_PAUSE)

# -----
# MAIN RUNNER
# -----

def run_demo() -> None:
    driver = make_driver()
    try:
        attempt(test_login, driver, expect_fail=True)
        attempt(test_admin_login, driver)
        attempt(creation_demo_user, driver)
        attempt(test_login, driver)
        attempt(test_new_serre, driver)
        attempt(test_new_bac, driver)
        attempt(test_new_culture, driver)
        attempt(test_logout, driver)
        print("\n? Demo complete.")
    except Exception as exc:
        print(f"\n? Demo failed: {exc}")
        raise
    finally:
        pause(SECTION_PAUSE)

```

```
driver.quit()
```

```
if __name__ == "__main__":  
    run_demo()
```

# Procédure de démo

Procédure de démo

# Script Selenium

---