**Plan de test Client MQTT : Projet Smart Territories**

**1- Identification du test**

Tests\_ClientMQTT\_SmartTerritories

**2- Référence du module testé**

Ces tests seront effectués sur les méthodes de la classe CClientMQTT, on travaillera sur la réception des données ainsi que les possibilités de création d’une nouvelle instance de la classe.

**3- Objectif du test**

Ce test vérifiera la bonne réception des données, ainsi que les différents codes d’erreur lors de leur validation. Les différents scénarios seront validés.

**4- Procédure du test**

*Test 1.1 : Instanciation de client MQTT par défaut*

Ce test correspond au constructeur de la classe, on instancie un nouvel objet du client MQTT avec les valeurs proposées par défaut ainsi qu’un analyseur lambda (classe CAnalyse non incluse).

En suivant la procédure de la partie 6 : Moyens à mettre en œuvre, ainsi que l’annexe de fiche de test, on obtient les résultats suivants :

« Client MQTT instancié : CClientMQTT {  
 topicName: 'test/topic',  
 objCAnalyse: {},  
 addrBroker: '127.0.0.1',  
 port: '1883',  
 client: null  
} »

On remarque que « client » est à « null » car il n’est pas encore connecté (méthode \_connect).

*Test 1.2 : Client MQTT avec données personnalisées*

Ce test correspond au constructeur de la classe, on instancie un nouvel objet de ClientMQTT, cette fois-ci avec des données personnalisées. L’analyseur sera toujours lambda.

En suivant la partie 6 du plan de test ainsi que l’annexe, on obtient le résultat suivant :

« Client MQTT personnalisé instancié : CClientMQTT {  
 topicName: 'custom/topic',  
 objCAnalyse: {},  
 addrBroker: '192.168.1.100',  
 port: '1884',  
 client: null  
} »

Test 2 : Connexion au broker MQTT

Ce test correspond à la méthode de connexion de la classe (\_connect) permettant de se connecter au broker MQTT.

En suivant la procédure on obtient le résultat suivant :

« Connecté au broker MQTT  
Abonné au topic : test/topic »

Test 3 : Réception de message MQTT

Ce test correspond à la méthode de réception de messages MQTT. On prépare donc il client MQTT puis on lui envoie un message à l’aide du topic choisi et du broker mosquitto (ici en localhost) puis après avoir lancé le programme, copier-coller la ligne de broker mosquitto (précisée en dessous du code) dans un autre terminal. Ainsi on envoie des données dans un format accepté (JSON) et les données sont reçues par le client :

« Message reçu : {  
 Capteur: 'Roseraie',  
 TypeDeDonnee: 'Temperature',  
 Date: '2025/05/19 15',  
 Valeur: 23.4  
} »

Test 4 : Message mal formaté

On ne précise pas que les données doivent être reçues en JSON, une erreur de format est vite arrivée. Il faut donc vérifier ce qui nous est retourné lors de cette erreur. On répète donc la même procédure que le test précédent, mais cette fois-ci le message MQTT envoyé n’est pas bon. On reçoit donc l’erreur suivante :

« Erreur de traitement du message : Unexpected token m in JSON at position 0 »

Le programme nous prévient qu’une erreur est survenue lors de la récupération du message.

**5- Résultats attendus**

Test 1.1 :

On créé un client MQTT par défaut, on s’attend donc dans ce test, à recevoir les informations prévues pour un client par défaut :

topicName (donné)  
 addrBroker : 127.0.0.1 (localhost par défaut)  
 port : 1883 (port MQTT par défaut)

Test 1.2:

Cette fois-ci, on créé un client MQTT avec des données personnalisées, le retour donné doit donc correspondre avec les informations inclues dans le constructeur (peu importe la demande de l’utilisateur)

Test 2 :

Pour ce test, on se connecte au broker MQTT choisi dans le constructeur, fait automatiquement dans le code en utilisant son url. On s’attend donc à recevoir le message qui nous valide la connexion au broker ainsi que le topic.

Test 3 :

On teste ici la réception de messages MQTT en format JSON. Les données n’ont pas de raison d’être refusées. On s’attend donc à recevoir les données reçues sans erreur, et toujours dans le bon format.

Test 4 :

On envoie des données dans le mauvais format, elles seront donc refusées par le code. On S’attend donc à ce qu’elles soient refusées avec le message d’erreur nous permettant de savoir ce qui ne va pas.

**6- Moyens à mettre en œuvre**

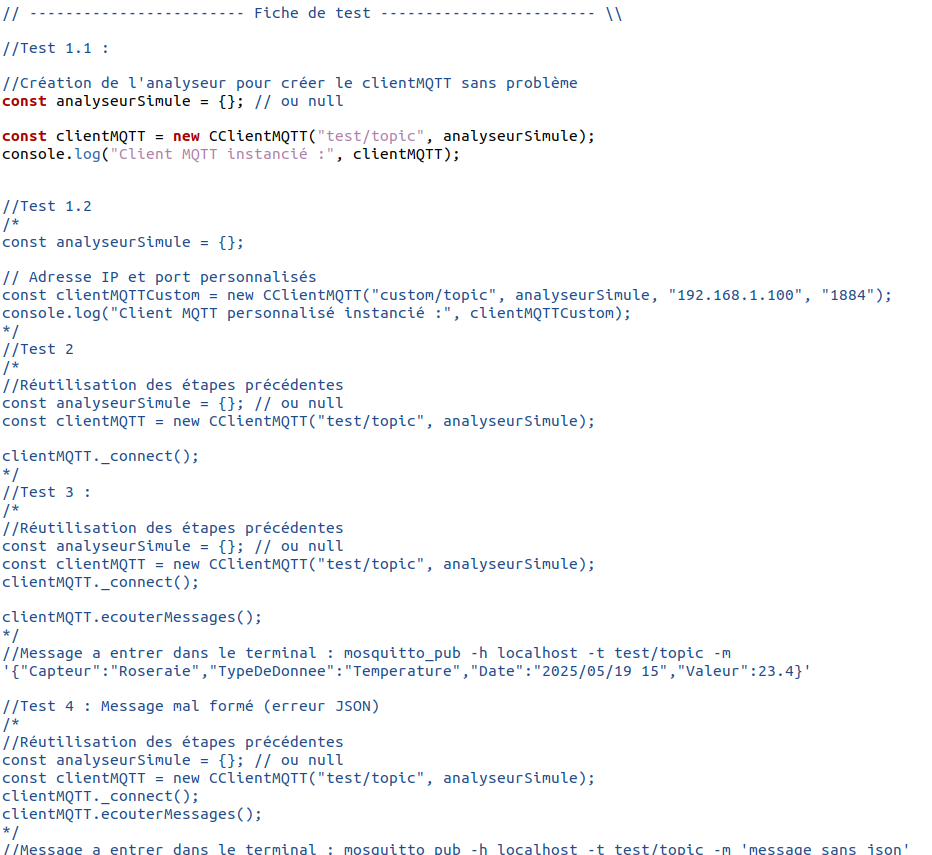
Pour effectuer ces tests il faudra se servir de la machine virtuelle servant de test au code.

Avant d’effectuer ces tests, il faudra démarrer le programme de test dans le terminal de la machine virtuelle. Il faudra se positionner dans le dossier : « Documents/API-ProgObjet/Tests » (cd Documents/API-ProgObjet/Tests) puis faire la commande : « node test-CClientMQTT.js » pour lancer le programme. Ensuite, on pourra se positionner dans un autre terminal et envoyer les données au client par le broker.

Dans le cas des tests, le broker est installé en localhost, ainsi on utilisera un broker directement positionné sur le serveur, cela nous permet de ne pas avoir besoin d’un accès à un réseau public. Il est tout de même possible, avec la configuration réseau nécessaire, de se connecter en passant par un réseau public.

Annexe : Fiche de test et programme modifié

Test 1.1 :



Test 1.2 :

Une image contenant texte, capture d’écran, document, Police

Le contenu généré par l’IA peut être incorrect.

Test 2 :

Une image contenant texte, capture d’écran, document, Police

Le contenu généré par l’IA peut être incorrect.

Test 3 :

Une image contenant texte, capture d’écran, document, Police

Le contenu généré par l’IA peut être incorrect.

Test 4 :

Une image contenant texte, capture d’écran, document, Police

Le contenu généré par l’IA peut être incorrect.