Documentation projet IOT :

**Publisher :**

variables :

* **Topic** = le topic de votre message
* **value** = le message publié sur le topic
* **id** = votre id **publisher unique**

**Connection au serveur mqtt :**

*const* publisher = mqtt.connect("mqtt://test.mosquitto.org", {clientId:"mqtt-publisher**id**"});

**Envoie d’un message sur un topic :**

publisher.publish("Hub\_**Topic**", **value**);

**Client :**

variables :

* **Topic** = le topic de votre message
* **id** = votre id **client unique**

**Connection au serveur mqtt :**

*const* client = mqtt.connect("mqtt://test.mosquitto.org", {clientId:"mqtt-client**id**"});

**Inscription au Hub (obligatoire) :**

client.publish("Hub\_Client-Subscribe", "**id**\_**Topic**");

**Abonnement au Hub pour recevoir les messages de mon topic :**

client.subscribe("Client**id**\_**Topic**");

**Valider la réception d’un message (obligatoire ou vous serrez automatique déconnecté) :**

client.publish("Hub\_Client-Response", **id**);

**Désinscription au Hub :**

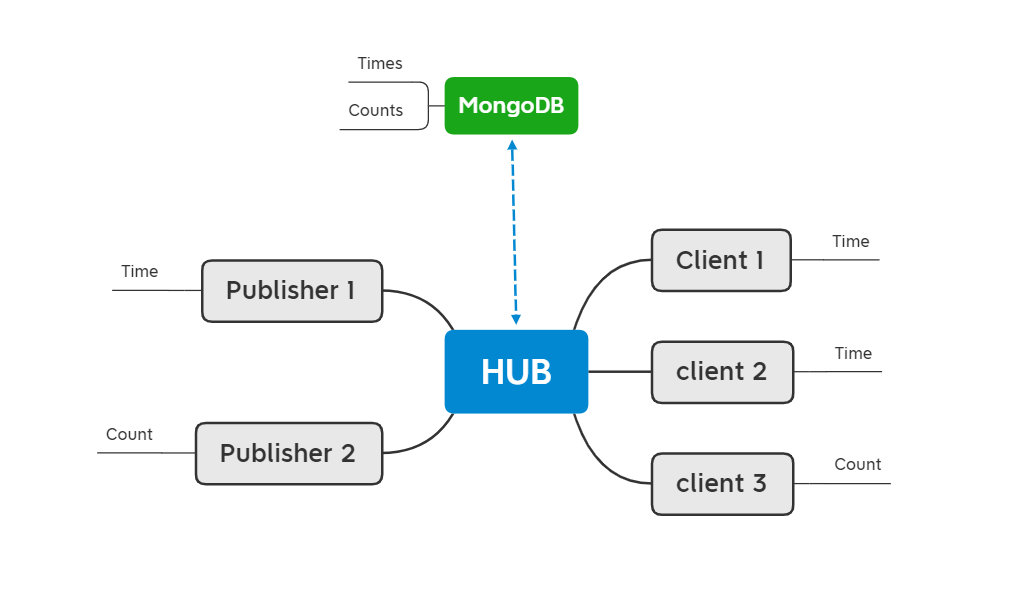
client.publish("Hub\_Client-UnSubscribe", **id**);

Descriptif approfondie projet IOT :

L'objectif de ce projet est de mettre en place un système de gestion de files d'attente des messages (similaire à RabbitMQ) en utilisant MQTT et MongoDB. Le système comprend deux publishers, un hub central et trois clients. Le hub agit comme un intermédiaire entre les publishers et les clients, stockant temporairement les messages dans une base de données MongoDB. Les clients peuvent recevoir les messages des différents topics par le hub, et une fois qu'un message est correctement reçu par un client, il est supprimé de la base de données par le hub.

#### **Architecture :**

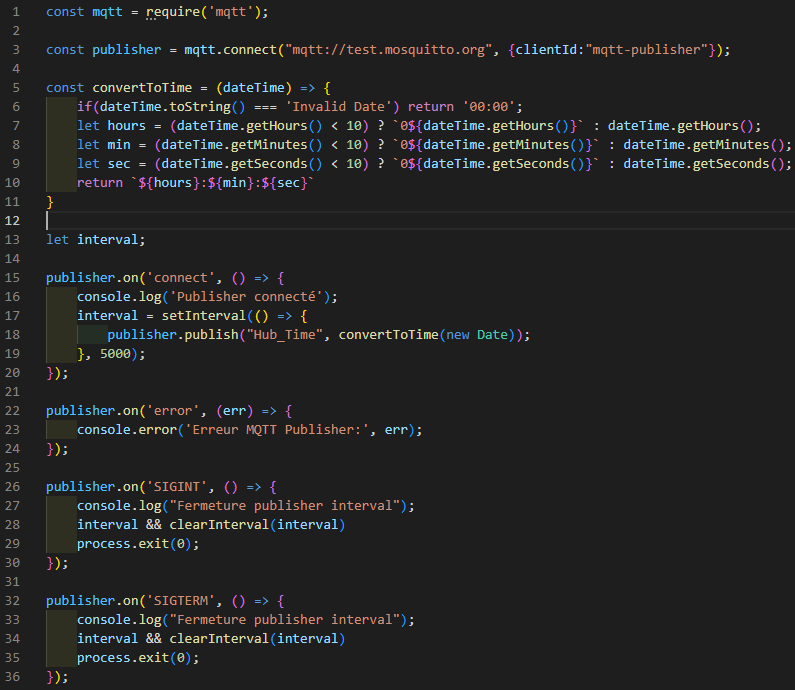
1. **Publishers** : Deux publishers publient des messages sur deux topics différents (Time et Count).
2. **Hub** : Reçoit les messages des publishers et les stocke dans MongoDB. Répond aux requêtes des clients avec les messages correspondants et supprime les messages dont la livraison est confirmée par le client.
3. **Clients** : Trois clients dont deux abonnés au même topic. Ils envoient des requêtes au hub pour récupérer les messages.
4. **MongoDB** : Permet de stocker temporairement les messages publiés par les 2 publishers.



#### 

#### **Publisher :**

Le code ci-dessous montre comment un publisher se connecte au broker MQTT, publie des messages à intervalles réguliers, et gère les erreurs et l'arrêt du processus.



La fonction **convertToTime** prend un objet Date en entrée et retourne une chaîne de caractères représentant l'heure sous le format HH:MM:SS.

Gestion des heures, minutes et secondes : Ajoute un zéro devant les heures, minutes et secondes si elles sont inférieures à 10 pour garantir un format à deux chiffres.

On va se servir de cette fonction pour générer la valeur que l’on va inclure dans notre message, sur le topic **Hub\_Time**.

**let interval** : Variable utilisée pour stocker l'identifiant de l'intervalle qui sera utilisé pour la publication périodique des messages.

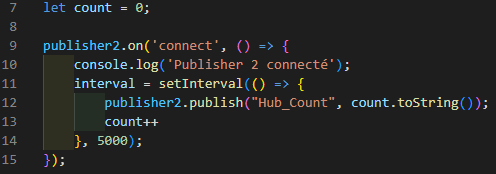
Avec l'événement **connect** comme son nom l’indique, on va se connecter au broker et envoyer un message toutes les 5 secondes, avec l’heure au format HH:MM:SS sur le topic Hub\_Time.

L'événement **error** permet d’envoyer un message en console en cas d’erreur lors de la connexion avec le client MQTT.

Les événements **SIGINT** et **SIGTERM**  sont déclenchés lorsque le processus reçoit les signaux **SIGINT** (Ctrl+C) ou **SIGTERM** (terminaison du processus), permettant notamment de supprimer l'intervalle de publication pour arrêter l'envoi des messages et de terminer le processus de manière “propre”.

**Publisher 2 :**

La différence avec le premier publisher se trouve sur la valeur envoyée dans le message, en effet ici on va publier un compteur qui s’incrémente à chaque message envoyé sur le topic **Hub\_Count**.



**Clients :**

Le code ci-dessous montre comment un client se connecte au broker MQTT, publie et s'abonne à des messages, et gère les erreurs et l'arrêt du processus.



Lors de la connexion, le client se connecte au broker MQTT en utilisant l'identifiant "mqtt-client1".

Une fois connecté, il publie un message sur le topic **Hub\_Client-Subscribe**. Ensuite, il s'abonne au topic **Client1\_Time** dans le but de recevoir des messages.

L'événement **message** est déclenché chaque fois que le client reçoit un message sur un topic auquel il est abonné. Si le message est reçu sur le topic **Client1\_Time** (topic sur lequel se trouve les différents messages attendu c’est à dire une heure précise) , le message est affiché dans la console et le client publie une réponse sur le topic **Hub\_Client-Response** avec la valeur **1**, pour indiquer qu’il a bien reçu le message, cela va permettre ensuite au **hub** de supprimer le message envoyé.

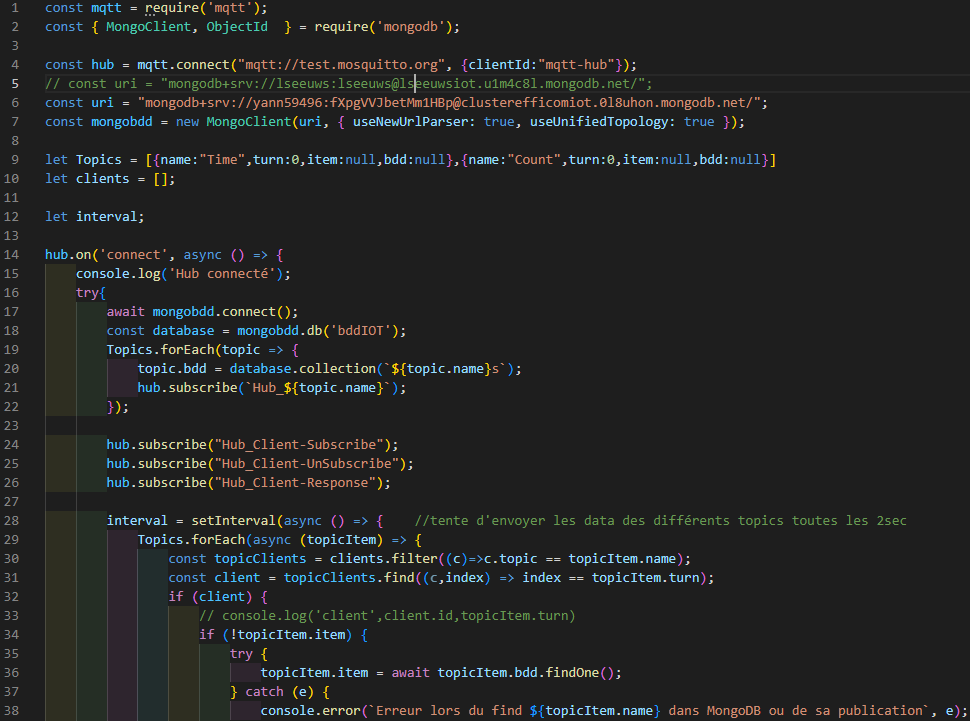
L'événement **error** permet d'envoyer un message en console en cas d'erreur lors de la connexion avec le client MQTT. Cela permet de diagnostiquer et de résoudre les problèmes de connexion ou de communication.

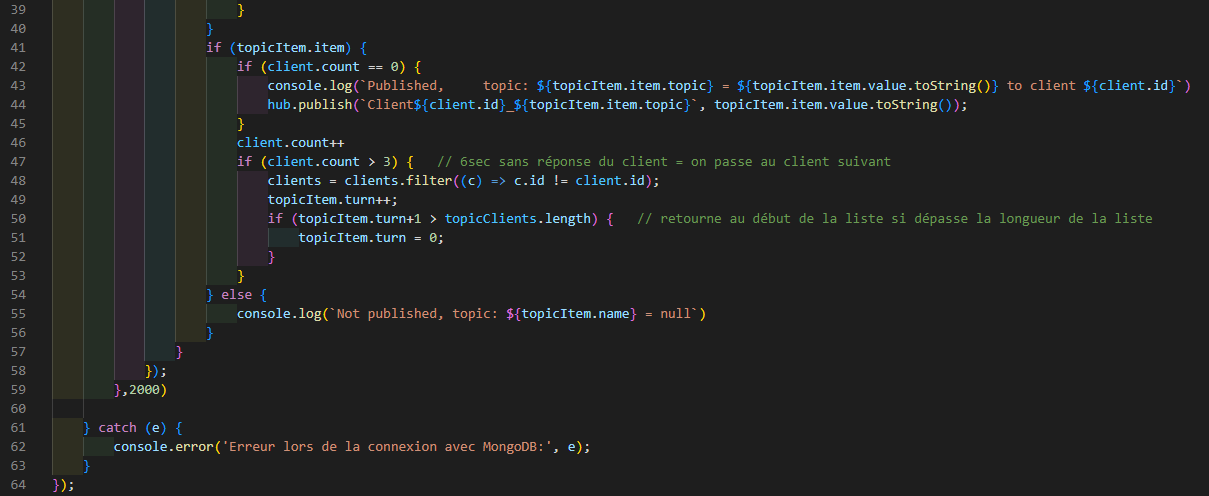
Les événements **SIGINT** et **SIGTERM** sont déclenchés lorsque le processus reçoit les signaux **SIGINT** (Ctrl+C) ou **SIGTERM** (terminaison du processus). Dans ces cas, le client publie un message sur le topic **Hub\_Client-UnSubscribe** indiquant qu'il se désabonne, puis termine le processus de manière propre.

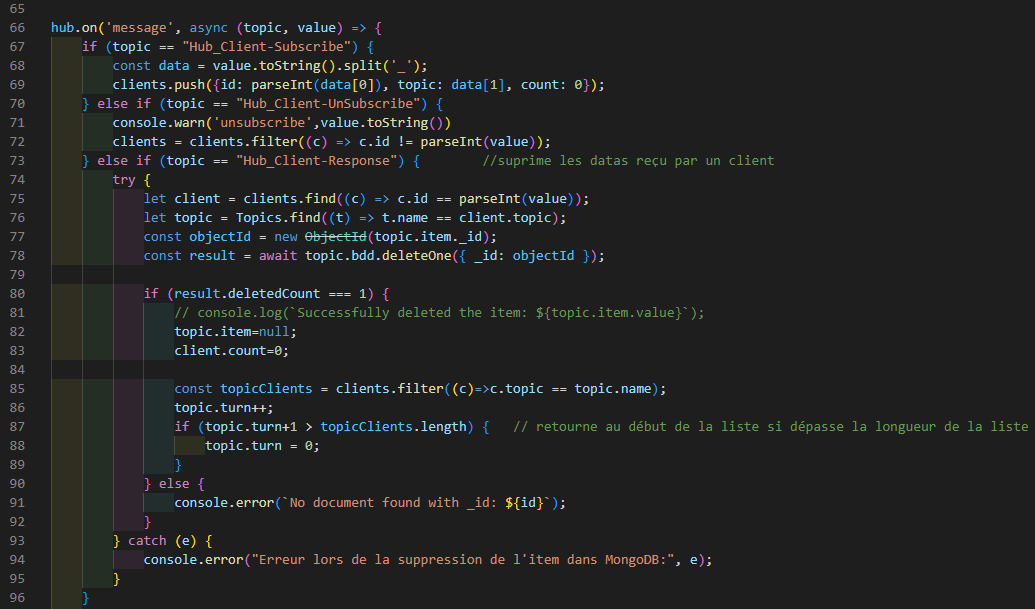
Autres clients : Il existe également deux autres clients dont un qui est abonné au topic **Client2\_Time** et un autre au topic **Client3\_Count**

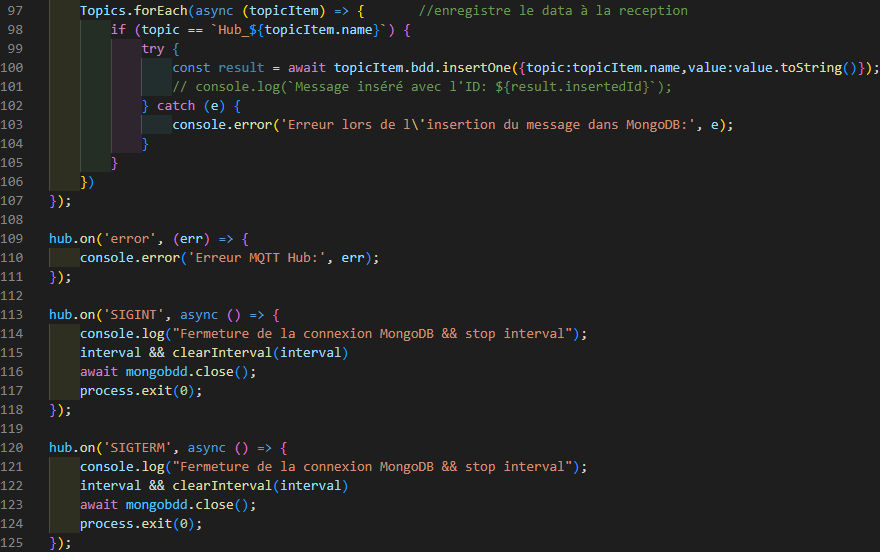
**Hub :**

Le code ci-dessous montre comment un hub se connecte au broker MQTT, interagit avec une base de données MongoDB, gère les abonnements et publications de messages, ainsi que les erreurs et l'arrêt du processus.









#### **Connexion au broker MQTT et à MongoDB**

Lors de la connexion, le hub se connecte au broker MQTT en utilisant l'identifiant mqtt-hub. Il se connecte également à une base de données MongoDB. Une fois connecté à MongoDB, le hub configure des collections pour chaque topic dans la base de données et s'abonne aux topics MQTT nécessaires.

#### **Gestion des topics et des clients**

Le hub maintient une liste de topics et de clients. Chaque topic est représenté par un objet contenant son nom, un compteur de tours (turn), l'item actuellement publié (item) et une référence à la collection MongoDB correspondante (bdd).

Les clients sont gérés dans une liste où chaque client est identifié par un ID, un topic auquel il est abonné et un compteur de tentatives (count). Lorsque le hub reçoit un message sur le topic **Hub\_Client-Subscribe**, il ajoute le client à la liste des clients. Lorsqu'il reçoit un message sur le topic **Hub\_Client-UnSubscribe**, il retire le client de la liste.

#### **Publication des messages**

Le hub tente d'envoyer des données des différents topics aux clients abonnés toutes les 2 secondes. Pour chaque topic, il sélectionne le client approprié en utilisant un compteur de tours. Si un item n'est pas encore chargé, il essaie de le récupérer depuis MongoDB. Si un client ne répond pas après 6 secondes le hub passe au client suivant et réinitialise le compteur.ç

#### **Réception des messages**

Lorsqu'un message est reçu sur un topic spécifique (**Hub\_Client-Subscribe**, **Hub\_Client-UnSubscribe** ou **Hub\_Client-Response**), le hub traite ce message en conséquence :

* Pour les abonnements, il ajoute le client à la liste.
* Pour les désabonnements, il retire le client de la liste.
* Pour les réponses des clients, il supprime l'item correspondant de MongoDB et réinitialise les compteurs.

#### **Enregistrement des messages**

Lorsque le hub reçoit un message sur un topic de données (par exemple, **Hub\_Time**), il enregistre ce message dans la collection MongoDB correspondante.

#### **Gestion des erreurs et de l'arrêt**

L'événement **error** permet d'envoyer un message en console en cas d'erreur lors de la connexion avec le client MQTT. Cela permet de diagnostiquer et de résoudre les problèmes de connexion ou de communication.

Les événements **SIGINT** et **SIGTERM** sont déclenchés lorsque le processus reçoit les signaux **SIGINT** (Ctrl+C) ou **SIGTERM** (terminaison du processus). Dans ces cas, le hub arrête l'intervalle de publication, ferme la connexion MongoDB et termine le processus de manière propre.

### 

### **Dépendances :**

**mqtt** et **mongodb**