UNIVERSITÉ
DE REIMS
CHAMPAGNE-ARDENNE

Système de Publisher/Subscriber avec Mosquitto en Java

Compte rendu TP5 INFO502

Élève :
Moussa TAYEB NEMICHE

Enseignants :
OLIVIER FLAUZAC
GEOFFREY WILHELM

Table des matières

Introduction

Ce rapport présente l'utilisation du protocole MQTT pour la gestion de la communication dans des applications distribuées. en implémentatn l'exemple d'un jeu de Poker Hold'em, on montre comment ce protocole permet de gérer efficacement les échanges de données entre les joueurs et les tables.

1 Connexion simple entre 2 acteur

1.1 Le Publisher

Le ***Publisher*** est responsable de la publication des messages sur un topic spécifique. Le code du *Publisher* utilise la bibliothèque Eclipse Paho pour gérer les connexions MQTT. Le rôle principal de cette classe est de se connecter à un broker MQTT, publier un message sur un topic donné, puis se déconnecter.

- **Connexion au broker MQTT** : La connexion est établie à l'aide de l'URL du broker et d'un identifiant de client. Dans notre cas, le broker est exécuté localement à l'adresse `tcp://127.0.0.1:1883`.
- **Création et publication du message** : Le message à publier est une chaîne de texte ("Bonjour, MQTT !") qui est convertie en un objet `MqttMessage`. Ce message est ensuite publié sur le topic `INFO0502`.
- **Déconnexion** : Après avoir publié le message, le client MQTT se déconnecte proprement et ferme la connexion.

Le ***Subscriber*** est responsable de la souscription à un topic spécifique et de la réception des messages publiés sur ce topic. Ce code utilise également la bibliothèque Eclipse Paho pour établir la connexion avec le broker MQTT et gérer la réception des messages.

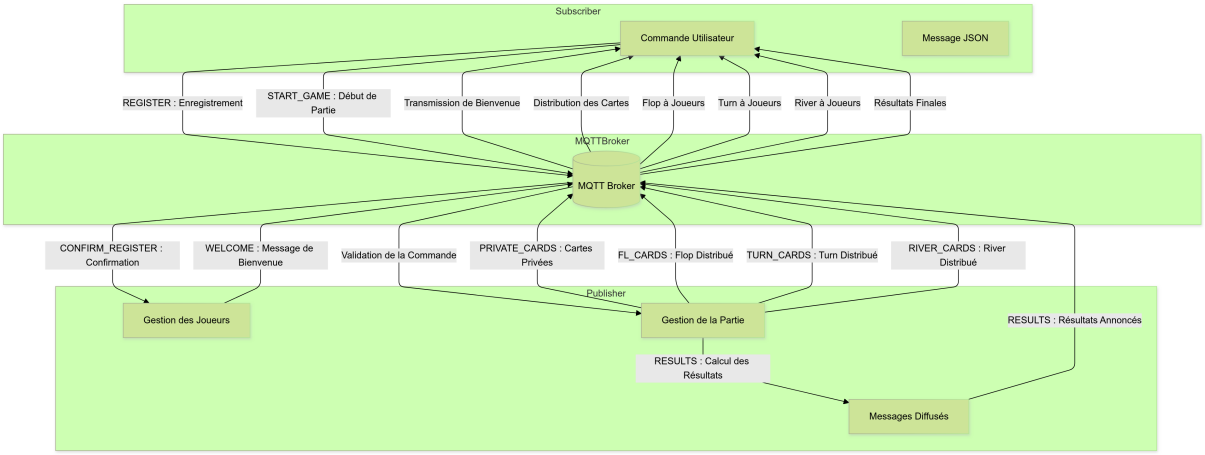
- **Connexion au broker MQTT** : Comme dans le Publisher, la connexion est établie avec le même broker à l'adresse `tcp://127.0.0.1:1883`. Le client MQTT utilise un identifiant différent pour la session, à savoir `subscriber`.
- **Souscription au topic** : Après la connexion, le client MQTT se souscrit au topic `INFO0502`, ce qui signifie qu'il commencera à recevoir tous les messages publiés sur ce topic.
- **Réception du message** : Dès qu'un message est publié sur le topic, la méthode `messageArrived` est appelée. Cette méthode affiche le contenu du message reçu.
- **Gestion des erreurs et pertes de connexion** : Le *Subscriber* inclut également des méthodes pour gérer la perte de connexion (`connectionLost`) et la confirmation de livraison des messages (`deliveryComplete`).

2 Gestion d'une seule table de pocker hold'em

2.1 Protocole MQTT

Le protocole MQTT avec mosquitto a étè utilisé pour son efficacité et sa légèreté dans les communications entre les différent acteur de cette application, c'est a dire les joueurs au sein d'une table de pocker :

- Les messages sont échangé rapidement et efficacement (sans perte) entre les différents acteurs.
- La gestion des connexions réseau instable grace aux mécanisme de reconnexion.
- MQTT utilise une architecture basé sur des topics, ce qui facilite la gestion des message ciblés, (ex : table, inscription, cartes, ect..)



2.2 Bibliothèques utilisées

- **Eclipse Paho** : Utilisée pour implémenter les clients MQTT (Publisher et Subscriber).
- **Gson** : pour sérialiser et désérialiser les messages JSON d'une façon simple et efficace.
- **Java Collections** : Les collections concurrentes (e.g., `ConcurrentHashMap`) permettent de gérer les données partagées de manière sûre entre threads.

2.3 Composition des classes

2.3.1 `Subscriber`

La classe `Subscriber` représente un joueur qui interagit avec le Publisher via des commandes MQTT. Voici les principales fonctionnalités :

- **Inscription au jeu** : Envoie un message de type `REGISTER` avec le nom du joueur.
- **Début de partie** : Permet de signaler de commencer une partie via une commande `START`.

- **Réception des messages** : Abonne le joueur à un topic personnel pour recevoir les informations.
- **Gestion des commandes utilisateur** : Un `Scanner` permet de lire les commandes depuis la console.

2.3.2 `Publisher`

La classe `Publisher` agit comme un serveur central qui :

- **Gère l'inscription des joueurs** : Les joueurs sont enregistrés dans une `ConcurrentHashMap` associant le nom des joueurs à leurs topics.
- **Gère le déroulement de la partie** :
  - Démarre une nouvelle partie avec au moins deux joueurs.
  - Distribue les cartes privées et les cartes communes (Flop, Turn, River).
  - Calcule les résultats et détermine le gagnant.
- **Diffuse les messages** : Utilise des topics pour envoyer des informations spécifiques ou diffuser des annonces générales.

2.3.3 `MessagePayload`

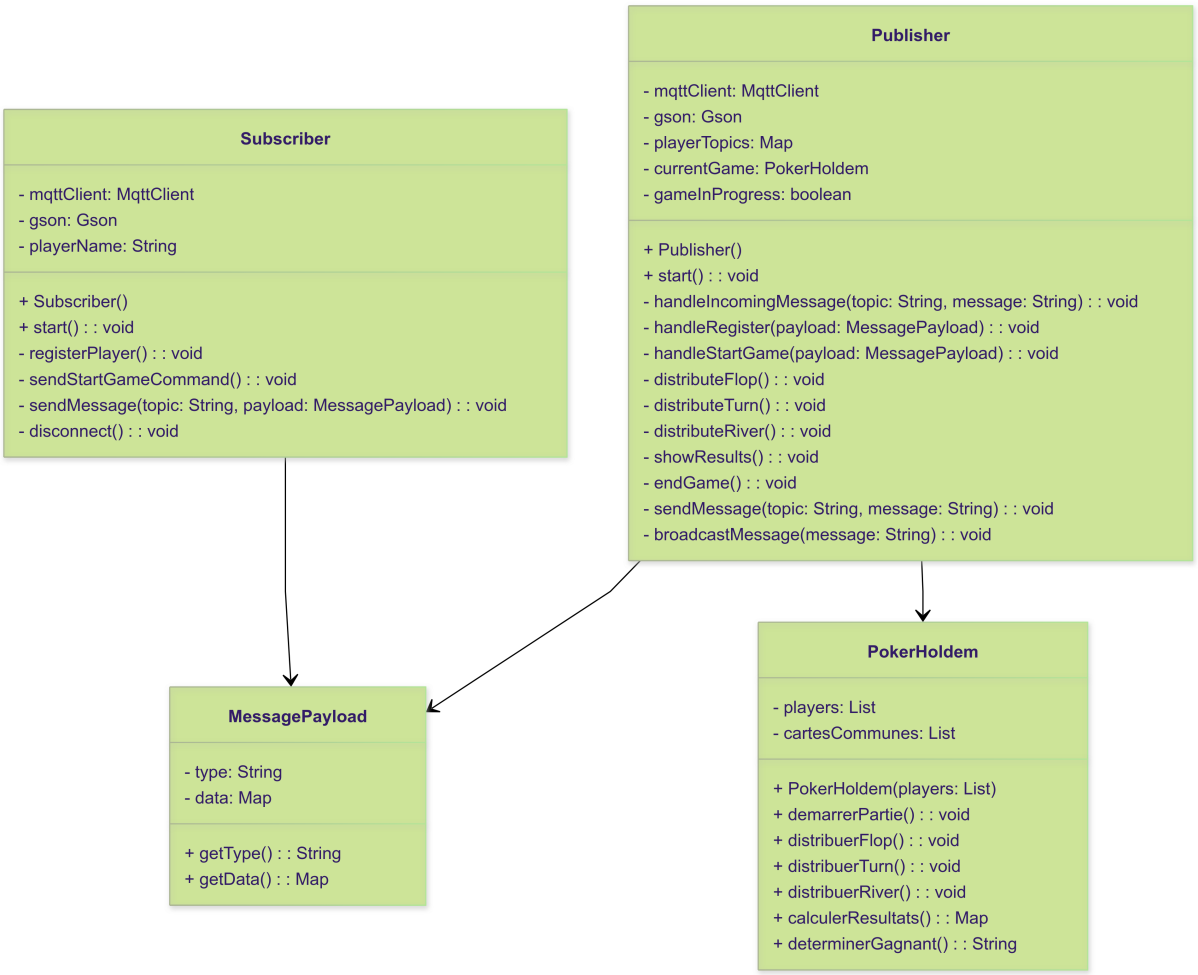
Cette classe est une structure de données pour encapsuler les messages échangés. Elle inclut :

- `type` : Spécifie le type de message (e.g., `REGISTER`, `START_GAME`).
- `data` : Contient les données supplémentaires, comme le nom du joueur ou les résultats de la partie.

2.3.4 `PokerHoldem`

La classe `PokerHoldem` représente la logique de gestion du jeu de poker. Elle :

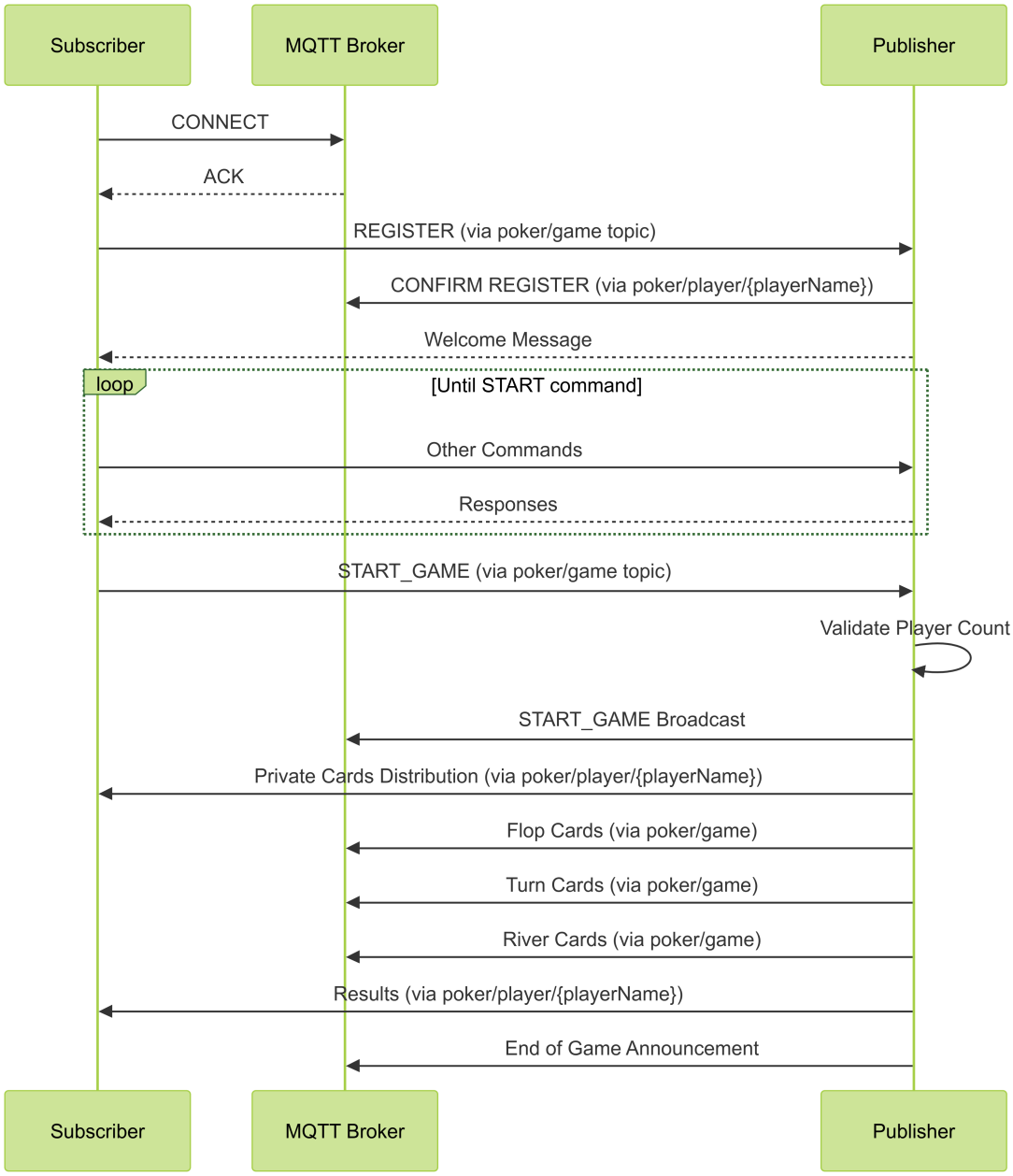
- Initialise les joueurs et leurs mains de départ.
- Gère la distribution des cartes communes.
- Fournit des méthodes pour calculer les résultats et déterminer le gagnant.



2.4 Interactions entre les modules

2.4.1 Flux de communication

1. Lorsqu'un joueur lance le client (`Subscriber`), il envoie un message de type `REGISTER` au serveur via le topic `poker/game`.
2. Le serveur (`Publisher`) enregistre le joueur et lui envoie un message de confirmation via un topic dédié (e.g., `poker/player/nom`).
3. Une fois que tous les joueurs sont enregistrés, un joueur peut envoyer la commande `START` pour démarrer la partie.
4. Le serveur distribue les cartes et diffuse les mises à jour du jeu via les topics appropriés.
5. Le déroulement se termine par l'envoi des résultats et l'annonce du gagnant.



2.4.2 Gestion des erreurs et reconnexion

- **Reconnexion automatique** : Gérée par `MqttConnectOptions` pour le serveur et `connectionLost` pour les clients.
- **Validation des messages** : Utilisation de Gson pour s'assurer que les messages reçus respectent le format JSON attendu.
- **Détection des doublons** : Vérification qu'un joueur ne peut pas s'inscrire deux fois.

3 Gestion de plusieurs tables et joueurs

3.1 Architecture du Système

L'architecture repose sur le protocole MQTT, un système de publication/abonnement léger adapté aux communications en temps réel. Voici les composants principaux :

3.1.1 Serveur MQTT (`Publisher`)

Le serveur gère :

- La création et la suppression des tables.
- La gestion des joueurs, y compris l'attribution des rôles et la diffusion des événements.
- Le démarrage et la progression des parties.

3.1.2 Client Joueur (`Subscriber`)

Chaque joueur utilise un client pour interagir avec le système :

- Rejoindre ou quitter une table.
- Démarrer une partie (pour l'administrateur).
- Recevoir les messages en temps réel concernant l'état des tables et les actions en cours.

3.1.3 Communications via MQTT

Les sujets MQTT utilisés sont :

- `poker/game/#` : Pour les événements généraux liés aux parties.
- `poker/game/table/<table_id>` : Pour les messages spécifiques à une table.
- `poker/player/<player_name>` : Pour les messages privés aux joueurs.

3.2 Fonctionnalités Implémentées

3.2.1 Serveur (`Publisher`)

Le serveur implémente les fonctionnalités suivantes :

1. **Création d'une table** : Un joueur peut créer une nouvelle table, devenant son administrateur.
2. **Rejoindre une table** : Les joueurs peuvent rejoindre des tables existantes.
3. **Lister les tables** : Le serveur diffuse la liste des tables disponibles.
4. **Démarrer une partie** : Une partie peut être démarrée par l'administrateur d'une table, à condition qu'il y ait au moins deux joueurs.
5. **Fermeture d'une table** : L'administrateur peut fermer une table, expulsant tous les joueurs.

3.2.2 Client Joueur (`Subscriber`)

Le client joueur offre une interface console pour :

- Créer ou rejoindre des tables.
- Voir la liste des tables disponibles.
- Recevoir des messages en temps réel sur les actions de jeu.
- Quitter ou fermer une table.

3.2.3 Gestion des Parties

Le serveur simule une partie de Texas Hold'em avec les étapes suivantes :

1. Distribution des cartes privées.
2. Révélation des cartes communes (*flop*, *turn*, *river*).
3. Calcul des résultats et annonce du gagnant.

3.3 Exemple de Communication MQTT

3.3.1 Création d'une Table

Le joueur publie un message sur le sujet `poker/game` :

```
{
    "type": "CREATE_TABLE",
    "data": {
        "player": "Alice"
    }
}
```

Le serveur répond par un message personnel sur `poker/player/Alice` :
Table créée avec succès. Vous êtes l'administrateur de la table <table$_id$ >

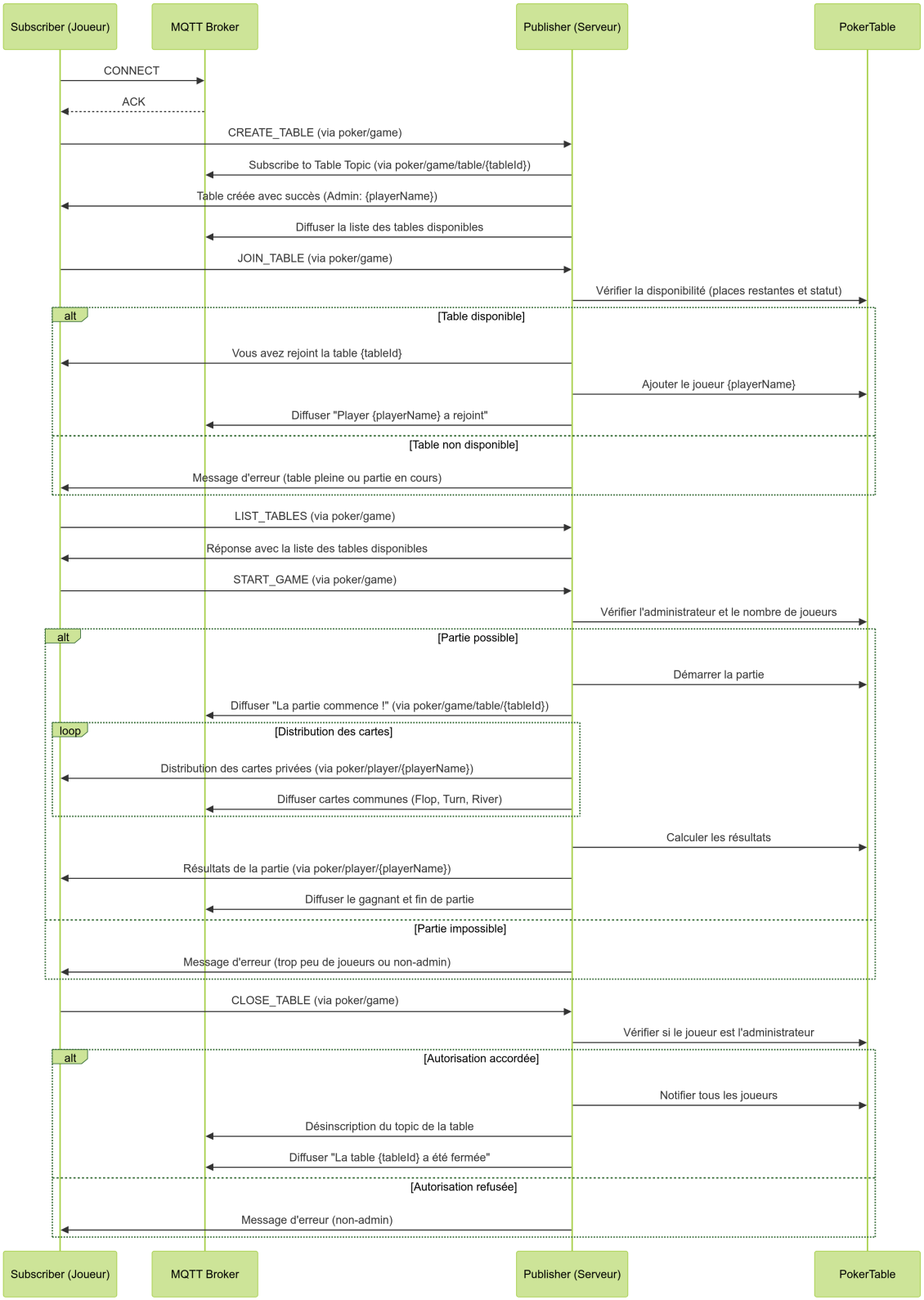
3.3.2 Rejoindre une Table

Le joueur publie un message sur `poker/game` :

```
{
    "type": "JOIN_TABLE",
    "data": {
        "player": "Bob",
        "tableId": "<table_id>"
    }
}
```

Le serveur diffuse un message à tous les joueurs de la table :

```
Bob a rejoint la table.
```


Subscriber (Joueur)
MQTT Broker
Publisher (Serveur)
PokerTable
CONNECT
ACK
CREATE_TABLE (via poker/game)
Subscribe to Table Topic (via poker/game/table/{tableId})
Table créée avec succès (Admin: {playerName})
Diffuser la liste des tables disponibles
JOIN_TABLE (via poker/game)
Vérifier la disponibilité (places restantes et statut)
alt
[Table disponible]
Vous avez rejoint la table {tableId}
Ajouter le joueur {playerName}
Diffuser "Player {playerName} a rejoint"
[Table non disponible]
Message d'erreur (table pleine ou partie en cours)
LIST_TABLES (via poker/game)
Réponse avec la liste des tables disponibles
START_GAME (via poker/game)
Vérifier l'administrateur et le nombre de joueurs
alt
[Partie possible]
Démarrer la partie
Diffuser "La partie commence !" (via poker/game/table/{tableId})
loop
[Distribution des cartes]
Distribution des cartes privées (via poker/player/{playerName})
Diffuser cartes communes (Flop, Turn, River)
Calculer les résultats
Résultats de la partie (via poker/player/{playerName})
Diffuser le gagnant et fin de partie
[Partie impossible]
Message d'erreur (trop peu de joueurs ou non-admin)
CLOSE_TABLE (via poker/game)
Vérifier si le joueur est l'administrateur
alt
[Autorisation accordée]
Notifier tous les joueurs
Désinscription du topic de la table
Diffuser "La table {tableId} a été fermée"
[Autorisation refusée]
Message d'erreur (non-admin)
Subscriber (Joueur)
MQTT Broker
Publisher (Serveur)
PokerTable

4 Structure du Code

Le système est divisé en deux classes principales :

- `Publisher` **: Responsable de la gestion des tables et des parties.**
- `Subscriber` **: Fournit une interface console pour les joueurs.**

4.1 Classe `Publisher`

Listing 1 – Extrait de la classe `Publisher`

```
public class Publisher {
    private final Map<String, PokerTable> tables = new
       ConcurrentHashMap<>();
    private final Map<String, String> playerTableMapping = new
       ConcurrentHashMap<>();

    public Publisher() throws MqttException {
        mqttClient = new MqttClient(MQTT_BROKER, MqttClient.
           generateClientId(), null);
        mqttClient.setCallback(new MqttCallbackAdapter());
        mqttClient.subscribe("poker/game/#");
    }

    private void handleCreateTable(MessagePayload payload) {
        String tableId = UUID.randomUUID().toString().substring(0,
           8);
        PokerTable table = new PokerTable(tableId, payload.getData
           ().get("player"));
        tables.put(tableId, table);
        playerTableMapping.put(payload.getData().get("player"),
           tableId);
    }
}
```

4.1.1 Classe `Subscriber`

Listing 2 – Extrait de la classe `Subscriber`

```
public class Subscriber {
    private final String playerName;
    private final MqttClient mqttClient;

    public Subscriber(String playerName) throws MqttException {
        this.playerName = playerName;
```

```
7           mqttClient = new MqttClient(MQTT_BROKER, "client-" +
                playerName, null);
8           mqttClient.setCallback(new MqttCallbackAdapter());
9       }
10
11      private void createTable() throws MqttException {
12          Map<String, String> data = new HashMap<>();
13          data.put("player", playerName);
14          publishMessage("poker/game", new MessagePayload("
                CREATE_TABLE", data));
15      }
16  }
```

5 Conclusion

Dans ce rapport, j'ai présenté trois applications du protocole MQTT, chacune avec ses spécificités, ce qui montre la capacité d'implémenter des applications efficaces en utilisant le protocole MQTT avec Mosquitto. Ce dernier permet de gérer des applications où plusieurs acteurs doivent être inclus et assurés. Dans le cadre de ce rapport, il s'agissait des joueurs et des tables dans un jeu de Poker Hold'em.