# RAPPORT PROGRAMMATION OBJET AVANCÉ MICRO-SERVICE

Corentin LEANDRE
Jerry CHAN

Sommaire :



# I - Introduction

Faire une application basée sur l'architecture microservice en TypeScript. On va réaliser le projet autour d’un jeu de puissance 4.

Les systèmes informatiques de nos jours évoluent constamment et doivent répondre en permanence à des problèmes liés à la charge et au trafic des usages. Aujourd’hui, les architectures traditionnelles d'applications monolithiques ou centralisées présentent plusieurs limitations qui impactent leur efficacité, leur maintenance et leur évolutivité. Ces problèmes sont tel que le Single Point Of Failure (SPOF) de l’ESB (Enterprise Service Bus). L’ESB, souvent utilisé comme middleware central, devient un point de défaillance critique. Si l’ESB échoue, l’ensemble du système peut être paralysé, entraînant des interruptions de service majeures. D’autres problèmes comme la scalabilité ou la maintenance interviennent et présentent de réels freins aux systèmes concernés.

Ainsi, l’architecture microservices apparaît ici comme solution permettant de pallier la plupart de ces problèmes en plus de garantir une certaine qualité de code.

# II - Fonctionnalités:

- Authentification (et distribution des jetons)
- Jeu (anonyme ou non)
- Vérification du jeton pour chaque transaction
- Matchmaking anonyme
- Création et gestion de salles de jeu

# III - Technologies utilisées:

## Docker

Pour simuler l'architecture microservices

## Socket.io

L'utilisation des sockets est stratégique car elle évite trop de complexité au niveau du routage lors de la réplication des  serveurs.

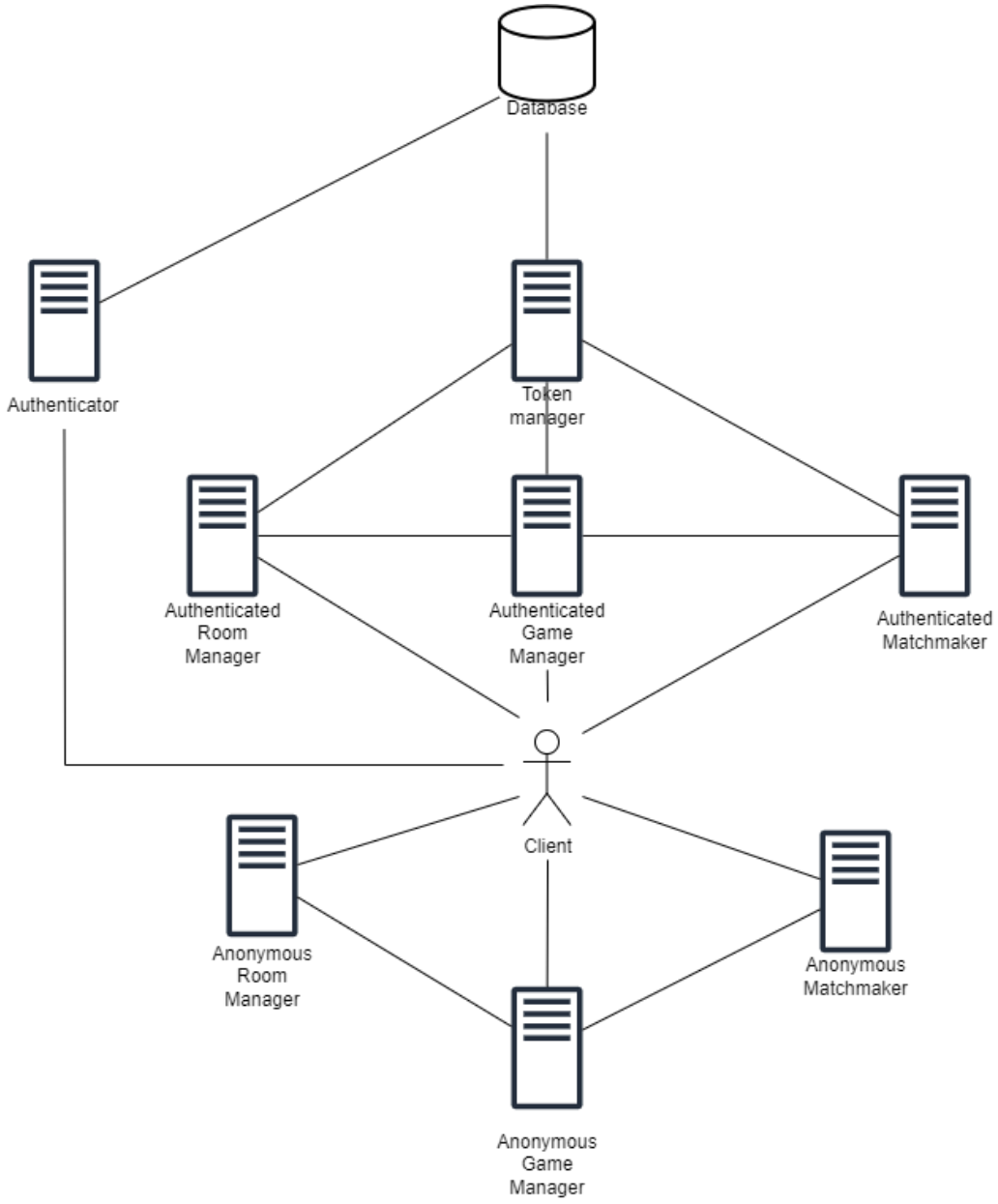
## Typescript & Webpack

Utilisé pour développer du javascript à grande ampleur, puis le compiler en javascript.
Webpack est utilisé pour compiler le Typescript côté client.
Typescript lui-même est utilisé pour le compiler côté serveur.

## Node

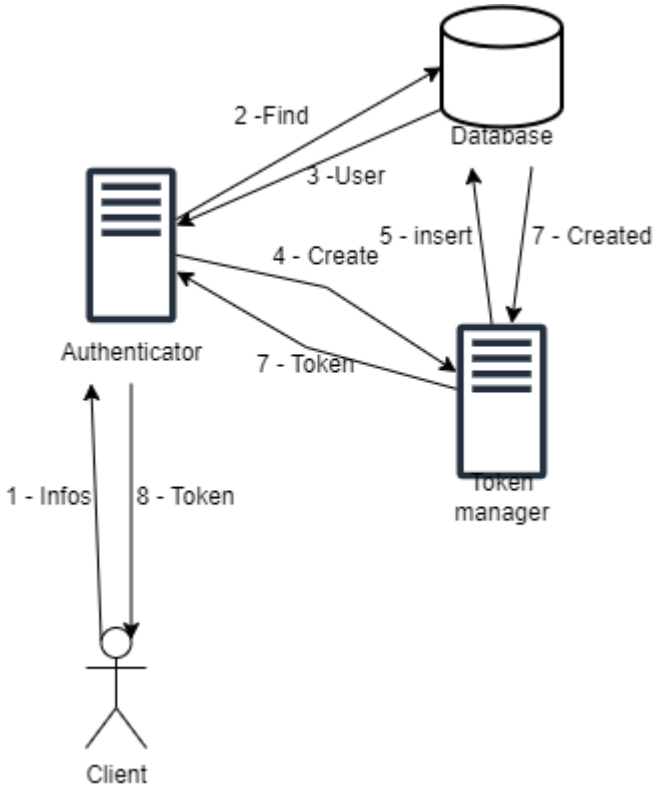
Pour faire le côté serveur en javascript.

# IV - Architecture Générale

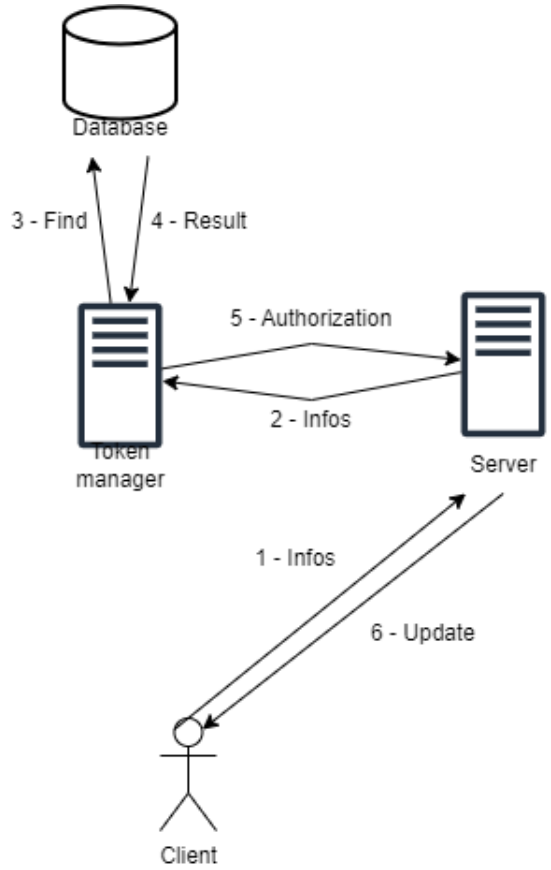



Le projet est découpé en module, chaque module fonctionnant indépendamment.

Schéma général pour la connexion

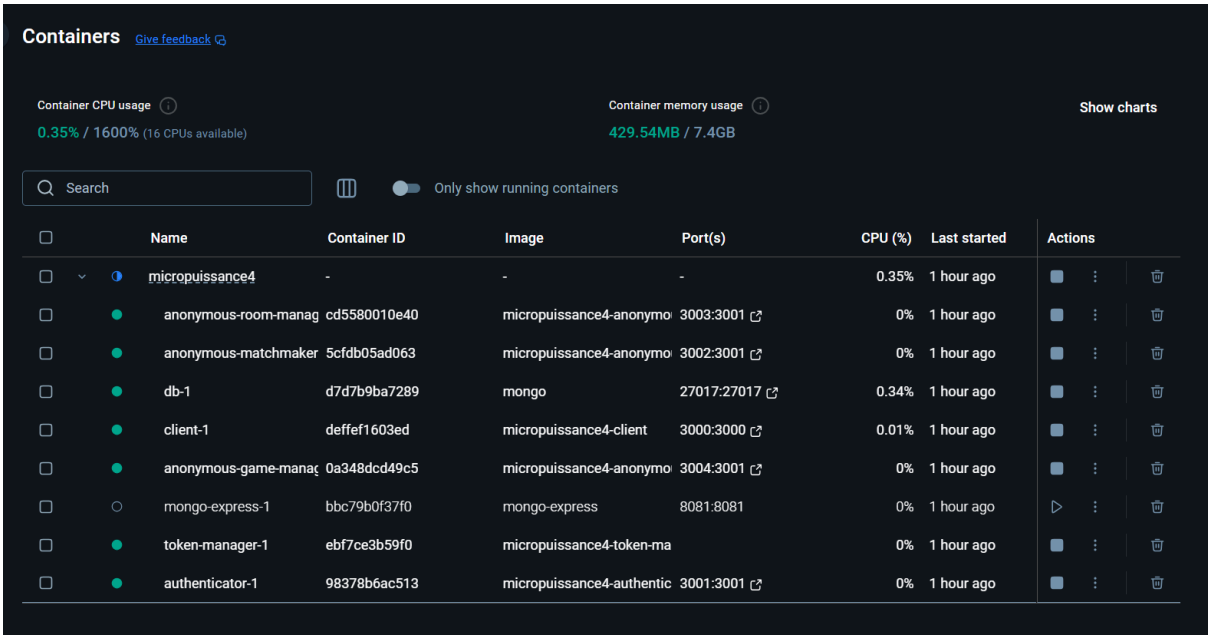


# Schémas général des transactions



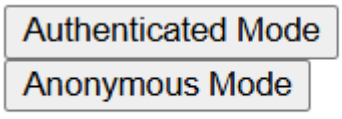
# V - Présentation des résultats

## Liste des containers sur docker



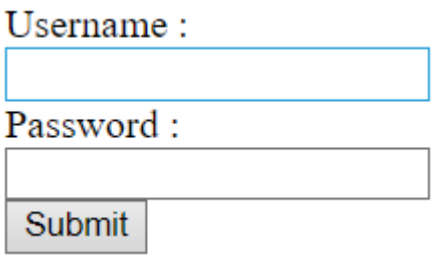

Ainsi, le système permet ainsi de démarrer une partie en mode anonyme. Le système d'authentification et le mode anonyme possèdent une seule occupation.

1ère page:



## Variante Authentifié (Authenticated):

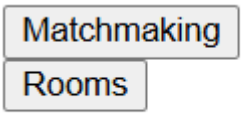
Renvoie vers une page de connexion



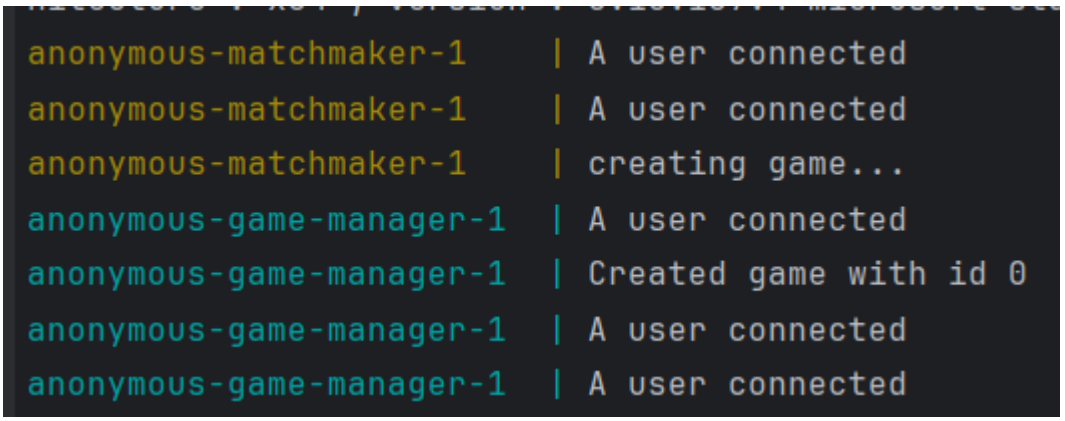

Cette variante utilise un jeton d'authentification lors de toutes les transactions pour les valider. À chaque transaction avec le serveur, si le jeton n'est pas valide, la transaction est ignorée.
Le reste des identique à la version anonyme, mais sur des serveurs séparés,d'où l'existence de versions anonymes et authentifiées des serveurs.

## Variante Anonyme (Anonymous):

renvoie vers la page de jeu

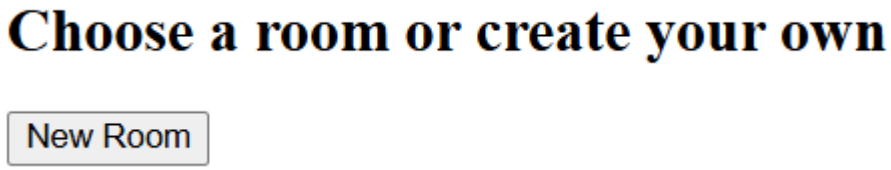



Matchmaking: Se met en attente d'un autre joueur. Si un autre joueur utilise le matchmaking, ils sont mis en contact et démarrent une partie.

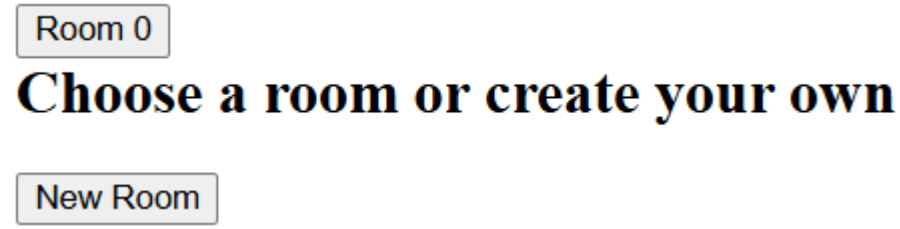



Les microservices anonymous-matchmaker et anonymous-game-manager vont entrer en contact et créer la partie.
Ici, on voit 2 utilisateurs se connecter au matchmaking.
Ensuite, le matchmaking crée la partie.
Pour cela, il se connecte au game manager et lui demande de créer la partie.
Celui-ci crée la partie.
Les 2 utilisateurs se connectent ensuite à la partie créée.

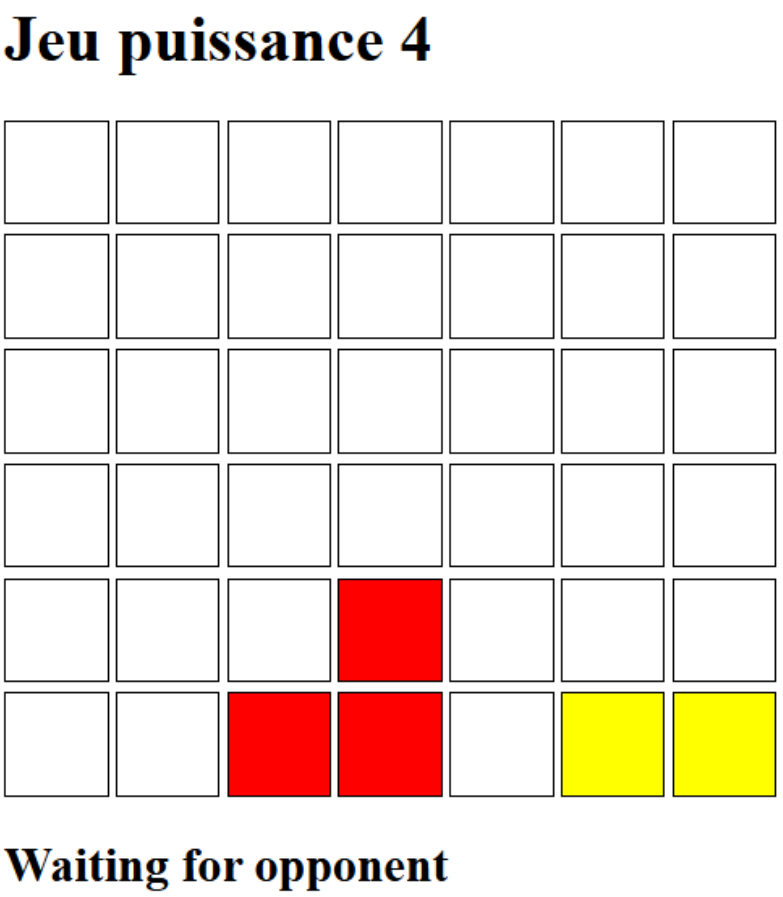
Rooms: Donne sur l'écran des salons disponibles. D'ici, on peut choisir de créer son propre salon ou se connecter à un des salons disponibles. Dès que 2 joueurs sont connectés au même salon, la partie démarre.




en cliquant sur New Room Le 1er joueur va créer une room.




Le 2eme joueur voir la room créé par le 1er joueur.




Le système utilise l’architecture microservice avec succès.

# VI - Conclusion

Le projet nous a permis d'apprendre comment fonctionne l’architecture microservice, pour ensuite nous permettre de concevoir la nôtre. Cela nous a également porté à réfléchir sur la communication entre microservices, et sur comment les rendre extensibles.

# VII - Points d’amélioration et perspectives

Pour aller plus loin, on pourrait essayer de simuler l'architecture des applications à grande échelle. On pourrait mettre en place de protection contre les attaques DDOS en bloquant les IPs qui apparaissent trop souvent.

On pourrait également développer un proxy pour faire le routage vers les différents serveurs, au lieu d’utiliser les ports. Il faudrait également utiliser un domaine au lieu d’utiliser localhost.

On devrait également implémenter un système pour s’inscrire, ce qui n’est pas possible pour l’instant, mais devrait être relativement simple.

On peut également implémenter un classement.