Synchronisation d'un lecteur vidéo avec chat

UE CALC

Nassim ABDELGHANI

29 décembre 2021

Table des matières

1 Introduction

Sur la plateforme de streaming Twitch, il y a la possibilité de faire une diffusion vidéo en direct, avec laquelle des spectateurs peuvent interagir par l'intermédiaire d'un chat. La plateforme a également un service de vidéo à la demande, qui rend disponible une diffusion terminée, ainsi que le chat associé. Cependant, ce service est une solution web (ou mobile).

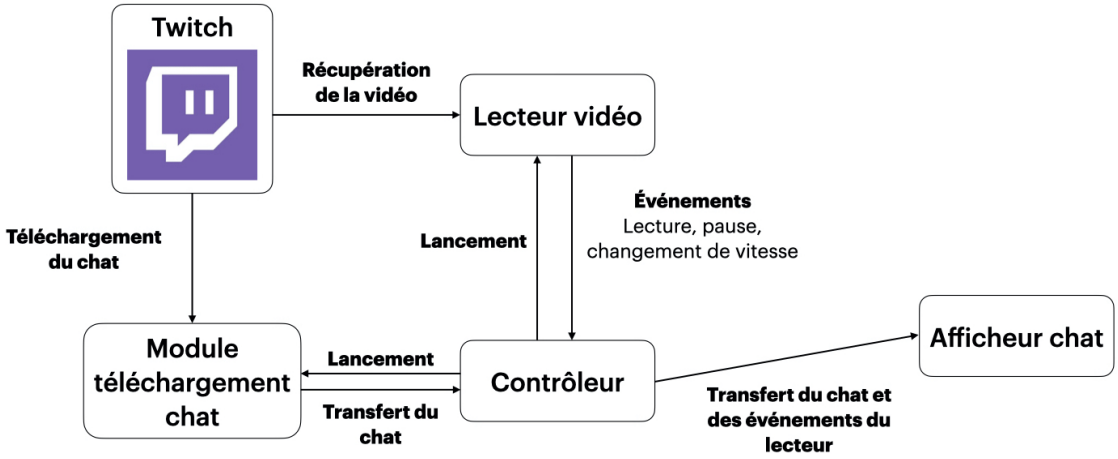
On pourrait vouloir utiliser un autre lecteur vidéo que celui d'un navigateur web, pour profiter de fonctionnalités que possède un lecteur vidéo dédié. Mais les solutions pour cela se font le plus souvent au sacrifice du chat. Il y a une possibilité qui est de générer un fichier vidéo à partir du chat, et de le lire côte à cote avec la vidéo. Mais cette dernière est longue, coûteuse en ressources et peu flexible.

Le but de ce projet est d'introduire un afficheur à la volée du chat d'une VOD Twitch, et de le synchroniser au lecteur vidéo.

2 Architecture logicielle

Il a été décidé d'avoir une sorte d'architecture microservice avec :
— un lecteur vidéo
— un module de téléchargement du chat
— un afficheur du chat
— un contrôleur

Le contrôleur se charge de démarrer un lecteur vidéo et le module de téléchargement du chat. Ce module se connecte à l'API de Twitch pour récupérer le chat et le transférer au contrôleur. Le lecteur récupère le flux vidéo de Twitch, et envoie des évènements liés à la lecture au contrôleur. Le contrôleur transfère le chat à l'afficheur, et les évènements du lecteur afin de la maintenir synchronisé.



2.1 Détails d'implémentation

2.1.1 Lecteur

J'utilise quotidiennement le lecteur MPV pour ses performances, et sa possibilité d'y utiliser des scripts et d'ainsi étendre ses fonctionnalités. Il intègre le module youtube-dl ce qui permet la lecture du contenu vidéo d'un nombre important de sites. Aussi ce lecteur fournit une interface JSON IPC permettant de l'intégrer à d'autres applications. C'est donc naturellement que j'ai construit le projet autour de ce lecteur, d'autant plus que c'est un logiciel multiplateforme, facilitant donc la portabilité du projet.

2.1.2 Module de téléchargement du chat

J'avais expérimenté avec le logiciel en ligne de commandes TwitchDownloader. Il permet de récupérer un fichier brut en JSON du chat, et d'en générer une vidéo comme présenté en introduction. J'ai donc réutilisé ce logiciel mais uniquement la partie de téléchargement, pour pouvoir m'occuper de l'affichage à la volée.

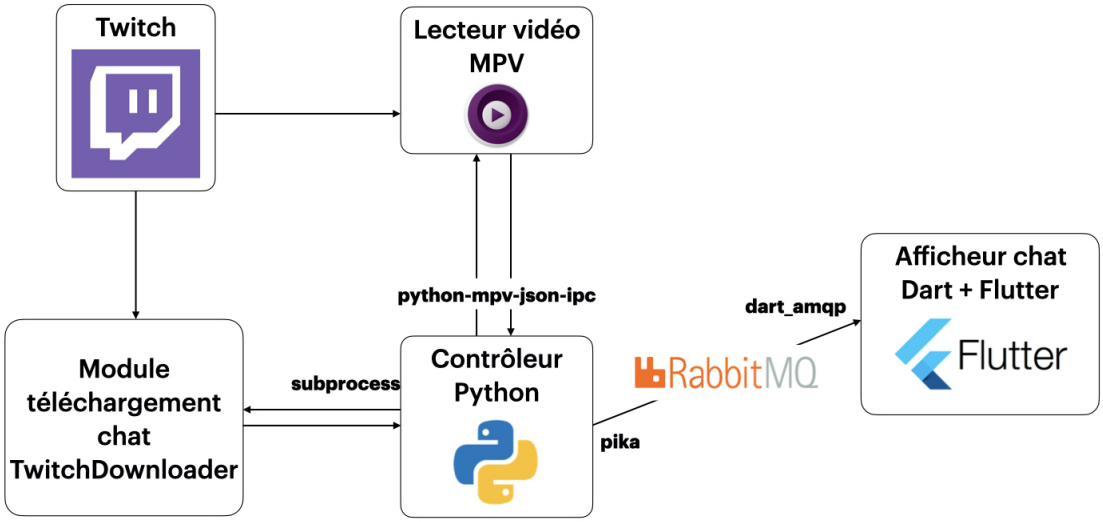
2.1.3 Afficheur chat

L'utilisation prévue de l'afficheur est sur la même machine que le lecteur avec en l'occurence le système d'exploitation macOS. Mais je ne voulais pas exclure une utilisation déportée sur un autre appareil, éventuellement mobile. J'ai donc choisi d'utiliser le framework Flutter avec le langage Dart. Ce framework est également multiplateforme, et promet de générer à partir d'un unique code source une application de bureau (Linux, macOS, Windows) ou bien mobile (iOS, Android). Il y a également une librairie dart_amqp implémentant en partie le protocole AMQP 0.9.1.

2.1.4 Contrôleur

Cette partie a été développée en Python.

— Une classe Player définit l'interface requise du lecteur vidéo (écoute d'évènements lecture, pause, changement de vitesse). Un quelconque lecteur peut donc être utilisé tant qu'il satisfait cette interface.
— Il existe une librairie python-mpv-jsonipc qui s'occupe déjà de communiquer avec le lecteur MPV, donc c'est ce que j'ai utilisé pour l'implémentation de la classe Player.
— Avec la librairie subprocess, on peut interagir avec le module de téléchargement.
— Enfin, avec la librairie pika, un client AMQP 0.9.1, il est possible de communiquer avec l'afficheur.



2.2 Utilisation de RabbitMQ

RabbitMQ a été mis en place pour la communication contrôleur-afficheur. Pour l'instant, tout se déroule sur une unique machine, donc le contrôleur ainsi que l'afficheur supposent qu'une instance de RabbitMQ tourne sur localhost sur le port standard 5672.

J'ai utilisé le mode topic pour sa flexibilité, avec les clés de routages suivantes :

— json : envoi du fichier brut JSON du chat

— sync.play : envoi de la position du lecteur en cas d'un évènement lecture
— sync.pause : envoi de la position du lecteur en cas d'un évènement pause
— sync.seek : envoi de la position du lecteur en cas de changement par une action utilisateur
— sync.speed : envoi de la nouvelle vitesse du lecteur en cas de changement
— sync.timer : envoi périodique de la position du lecteur pour régler d'éventuelles pertes de synchronisation
— exit : envoi d'un signal à la fermeture du lecteur

Par simplicité, seul le contrôleur est producteur, et l'afficheur est uniquement consommateur, on fait l'hypothèse que l'afficheur est actif lorsque le contrôleur envoie les messages. Côté afficheur une queue est dédiée à chacun de ces topics. Il était pratique pendant le développement d'avoir une queue écoutant les messages sync.* pour pouvoir débogguer.

L'utilisation de RabbitMQ a ici facilité la communication entre deux services. J'ai pu ainsi choisir les langages indépendamment, selon mes connaissances, ou le besoin du service. En l'occurence, j'ai choisi Python pour le contrôleur par facilité, et Dart + Flutter plus adapté au développement d'une interface graphique multiplateforme.

On peut aussi noter l'extensibilité d'une solution autour de RabbitMQ. L'ajout d'un fonctionnement RPC pour la récupération du fichier JSON du chat, permettrait par exemple qu'un afficheur puisse se connecter en cours de route. Et on pourrait imaginer avoir un paramètre pour le chemin d'accès à l'instance RabbitMQ. De cette manière, le lecteur et les afficheurs pourraient tourner sur des machines différentes. Par exemple un ordinateur connecté à un téléviseur ferait tourner le lecteur, et une tablette ferait tourner l'afficheur.

3 Conclusion

J'utilisais parfois la solution proposée en introduction dans le cas de diffusions courtes, mais pour une diffusion plus longue et/ou avec un chat actif, la génération d'une vidéo du chat pouvait prendre plusieurs heures, et faire abandonner l'utilisation du logiciel. En comparaison, la solution développée dans ce projet est seulement limitée au démarrage par le téléchargement du chat, mais cela ne dure que quelques minutes, ce qui est plus acceptable. Il pouvait aussi arriver que le chat et la vidéo ne soient pas exactement synchronisés. Avec MPV via FFMPEG, il était possible de régler un décalage d'un fichier par rapport à l'autre, mais ce n'était pas chose simple. Changer le décalage obligeait aussi à quitter puis relancer la vidéo, ne facilitant pas un réglage fin. Le développement d'un afficheur dédié facilite ce genre d'opérations sans le besoin de relancer la vidéo.

Malgré les cours, qui mentionnaient précisément les avantages d'une solution logicielle reposant sur RabbitMQ, cela était plutôt abstrait pour moi au début et j'avais besoin de concrètement voir ceux-ci à l'oeuvre. Par ailleurs, j'ai été un peu déstabilisé par la liberté donnée dans le choix du sujet. Mais finalement, cela m'a permis de toucher du doigt les avantages d'une telle solution, sur un projet que je me vois étendre et utiliser.