

# MewPipe

Documentation technique  
Développement

# Sommaire

- **Introduction**
- **Choix techniques**
- **Architecture logicielle**
- **Structure de données**
- **Designs patterns**
- **Ergonomie et expérience utilisateur**
- **Algorithmes**
- **Points particuliers**

# I. Introduction

Ce document contient l'ensemble des informations demandées par le sujet du projet ainsi que certains détails que nous avons jugé importants.

La documentation pour installer l'environnement nécessaire pour faire tourner MewPipe se trouve dans le document "Instructions de déploiement".

Ce document est destiné à un lectorat "Technique". Cela signifie que l'utilisation de termes techniques liés au domaine de l'informatique, et plus particulièrement du développement, sera fréquente.

A titre de rappel, notre groupe est composé des personnes suivantes :

- Yilmaz Fatma - 161794
- Quinquis Simon - 143643
- Huynh Eddy - 144074
- Evieux Jean-Baptiste - 144897
- Cholin Théodore - 145731
- Chevalier Alexis - 123750

## II. Choix techniques

Afin de commencer cette documentation, nous allons détailler nos choix techniques pour la réalisation de ce projet.

### Microsoft .NET

Nous avons choisi de réaliser ce projet en utilisant les solutions fournies par le framework .NET de Microsoft (ASP.NET, Applications console et Bibliothèques de classes).

La raison qui nous a poussé vers ce framework est que nous devons héberger la solution sur des serveurs Windows. Afin de maximiser l'intégration avec la plateforme, nous avons choisi d'utiliser les technologies proposées par l'éditeur de Windows, Microsoft.

De plus, l'équipe en charge du projet est habituée à ces outils, nous nous sommes également basés sur ce critère.

Enfin, vous trouverez facilement du support sur ces technologies, l'utilisation du framework .NET étant assez répandue.

### IIS

Afin d'héberger nos applications ASP.NET nous avons choisi Internet Information Services pour sa capacité à déployer des applications basées sur le framework .NET.

La documentation sur l'architecture reviendra plus en détail sur ce point.

### SQL Server

Nous utilisons SQL Server pour stocker la totalité de nos données dites "légères" (pas les fichiers physiques). Tout comme IIS, SQL Server est un standard pour les applications .NET.

Vous trouverez plus de détails dans la documentation architecture.

## **MongoDB**

Les fichiers vidéos sont eux stockés dans MongoDB (la partie GridFS). Nous avons choisi GridFS car nous avons déjà eu une expérience très satisfaisante avec ce dernier, il est facile d'étendre la capacité de stockage et l'intégration dans le projet était également très utile.

GridFS stocke les fichiers sous forme de petits morceaux que nous pouvons facilement récupérer/ajouter via des flux depuis le code.

D'autres détails sont mentionnés dans la documentation architecture.

## **RabbitMQ**

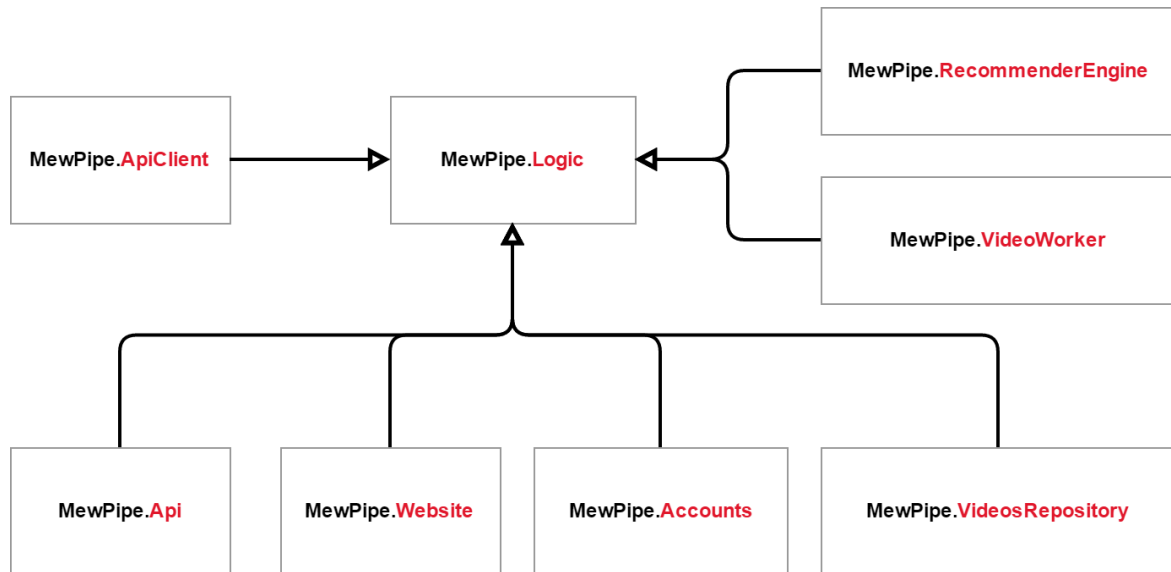
RabbitMQ est un système de file d'attente. Cela nous permet de gérer les traitements lourds de manière décentralisée dans MewPipe (Exemple: Les conversions vidéos).

Nous l'avons choisi car il est très reconnu dans son domaine, et qu'une partie de l'équipe connaissait cette technologie. De plus, des bibliothèques .NET étaient disponibles.

Encore une fois, les détails concernant la mise en place seront dans la documentation architecture.

### III. Architecture logicielle

Notre architecture logicielle est composée de divers projets au sein de la solution MewPipe, ces projets sont répartis de cette manière :



*Architecture logicielle de la solution MewPipe*

Voici à présent le détail projet par projet de ce schéma :

#### **MewPipe.Logic**

C'est le projet regroupant les services et les implémentations métiers, il ne dépend pas des autres projets et gère également les connexions aux sources de données.

Certaines logiques métiers qui n'ont pas pour but d'être partagées seront cependant trouvées dans les autres projets. Cette partie regroupe la couche de données et la couche métier.

#### **MewPipe.Website**

Notre site principal, l'interface qui permet d'utiliser MewPipe, pour les fonctions que vous nous avez demandées dans le sujet. Il utilise "ApiClient" pour toutes ses fonctionnalités.

## **MewPipe.VideoRepository**

Il s'agit de notre site qui gère les transferts de données, il permet :

- D'envoyer et de récupérer les fichiers vidéos
- De récupérer les images miniatures des vidéos

## **MewPipe.Accounts**

C'est notre site web qui gère l'authentification sur toute la solution. Il a plusieurs buts :

- Gérer les identités oAuth2 et les clés d'accès des utilisateurs ou applications externes
- Fournir le portail de connexion centralisé pour les utilisateurs de MewPipe
- Fournir les options de gestion de compte (récupération de mot de passe, mise à jour, suppression) dont l'utilisateur pourrait avoir besoin

Cette partie possède sa propre logique métier car la gestion de compte et des identités oAuth2 n'a pas pour but d'être disponible ailleurs dans le projet.

## **MewPipe.Api**

Il s'agit de notre site d'API, il contient les routes qui permettent aux développeurs d'utiliser pleinement MewPipe.

L'authentification sur l'API est sécurisée via oAuth2. L'API contient également une documentation visuelle sur les routes proposées.

## **MewPipe.ApiClient**

Consiste en une bibliothèque C# destinée à ouvrir l'utilisation de l'API. Un développeur peut utiliser ce projet pour travailler avec l'API de MewPipe.

Le client implémente toutes les routes disponibles sur le service, excepté la récupération/ajout de vidéos/miniatures.

## **MewPipe.RecommenderEngine**

C'est une application qui permet de déterminer les recommandations des vidéos. Une recommandation est une vidéo qui est susceptible d'intéresser un utilisateur qui vient de visionner une autre vidéo.

Le moteur fonctionne en deux modes :

- Reconstruction de la base : Il fait ceci en boucle, il prend toutes les vidéos et recalcule les recommandations afin d'avoir les valeurs les plus à jour possible.
- Traitement d'une nouvelle vidéo : Lorsqu'une vidéo est ajoutée, le moteur reçoit une notification via une file d'attente de "RabbitMQ" et traite cette vidéo pour lui trouver des semblables.

## **MewPipe.VideoWorker**

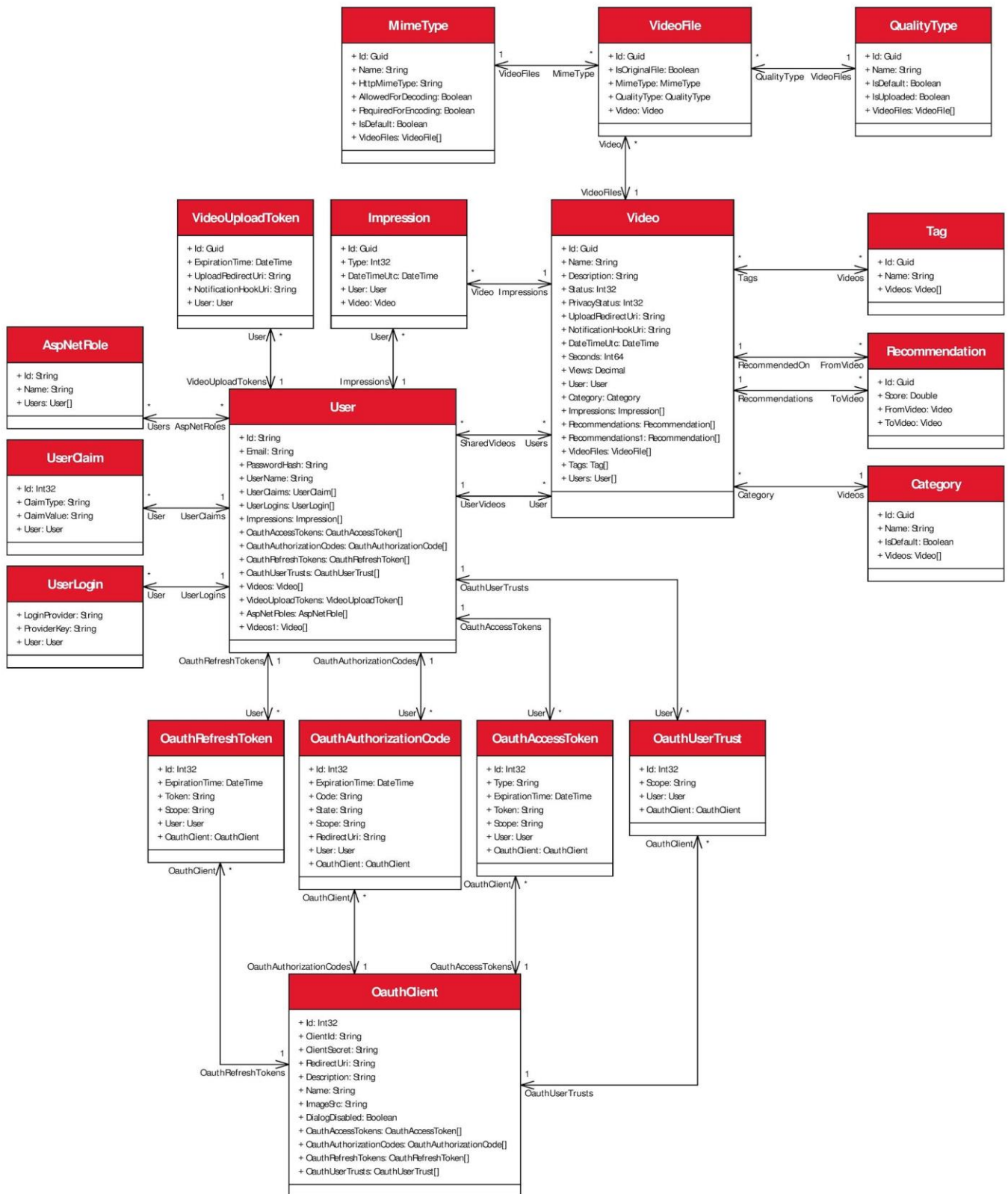
Notre moteur de traitement vidéo. Il est notifié de l'arrivée d'une nouvelle vidéo par une file d'attente et va ensuite la traiter. Le traitement se compose de trois étapes :

- Récupération des informations et du fichier envoyé par l'utilisateur
- Traitement de la vidéo pour récupérer une image miniature
- Conversion et stockage de la vidéo dans les formats MP4 et OGG et en 4 niveaux de qualité (moins si la vidéo est de mauvaise qualité)
- Mise à jour de la base de données



## IV. Structure de données

Le schéma UML suivant existe en version PDF (meilleure qualité) dans le dossier de rendu.



Comme l’UML sur la page précédente le montre, notre schéma de données est complexe, nous pouvons voir qu’il est centré sur l’utilisateur et ses vidéos. Nous allons décrire quelques points importants.

Il est à noter que les tables gérant les relations “Many to Many” ont été volontairement cachées pour des raisons évidentes de place.

## Utilisateurs

Les utilisateurs sont gérés via 4 tables: **User**, **AspNetRole**, **UserClaim** et **UserLogin**.

La table **User** stocke les données utilisateurs générales, la table **AspNetRole** est intégrée par défaut car nous utilisons le système d’identités utilisateurs d’ASP.NET.

La table **UserClaim** permet de générer des clés à usage unique destinées aux fonctionnalités utilisateur (comme la récupération du mot de passe par exemple).

Enfin, la table **UserLogin** permet de gérer les logins externes (OpenID, Facebook, etc...).

## OAuth 2.0

Le système d’authentification centralisée basé sur OAuth2.0 (<http://oauth.net/2>) utilise 5 tables, **OAuthRefreshToken**, **OAuthAuthorizationCode**, **OAuthAccessToken**, **OAuthUserTrust** et **OAuthClient**.

Ces tables stockent les données qui concernent le workflow OAuth2, nous ne rentrerons pas dans les détails car leurs noms sont explicites.

La table **OAuthUserTrust** en revanche, permet de mémoriser si un client est autorisé ou non par un utilisateur et avec quel scope.

## Vidéos

Les vidéos occupent la plus grande partie de la base de données avec 9 tables, **Video**, **VideoUploadToken**, **Impression**, **VideoFile**, **MimeType**, **QualityType**, **Tag**, **Recommendation** et **Category**.

La table **Video** stocke les informations logiques concernant la vidéo (son titre, sa description, les relations, etc).

Pour procéder à l’ajout d’une vidéo, l’utilisateur doit demander un token à l’API, ces tokens sont stockés dans la table **VideoUploadToken**.

L'information de "Like" et "Dislike" est appelée "Impression" et est stockée dans la table portant ce nom.

Une vidéo possède plusieurs fichiers (aux formats MP4 et OGG, dans les qualités 360p, 480p, 720p et 1080p). Nous stockons les références à ces fichiers dans **VideoFile** (les fichiers physiques sont stockés dans MongoDB). Nous trouvons également les tables **QualityType** et **MimeType** qui listent les formats et qualités disponibles pour les fichiers vidéos.

Les vidéos peuvent être "taguées" avec un système de mots clefs. Les mots clefs sont stockés dans la table **Tag**.

Notre système de recommandations tourne en arrière plan et vient mettre à jour la base de données périodiquement pour donner une liste de recommandations par vidéos. Elles sont stockées ici.

Enfin, une liste de catégories prédéfinies est disponible pour les vidéos, elles sont stockées dans la table **Category**.

## Fichiers physiques

Les fichiers physiques (fichiers vidéos et images miniatures) sont stockés dans MongoDB de manière très simple.

Nous avons deux bases de données, **Videos** et **Thumbnails**, qui récupèrent les fichiers, avec un nom unique composé de l'ID de la vidéo, de son format (video/mp4 par exemple) et de sa qualité (360p par exemple).

Les fichiers sont stockés dans GridFS (composant de MongoDB) sous forme de chunks (ou segments).

## Sessions

Dernière partie de notre système, nos sessions. Elles sont stockées dans un système séparé afin de permettre la mise en haute disponibilité.

Actuellement nous stockons les sessions dans MongoDB, dans une base nommée **Sessions**.

A terme lors du passage en production nous passerons vers SQL Server avec un système optimisé.

## **Données temporaires des recommandations**

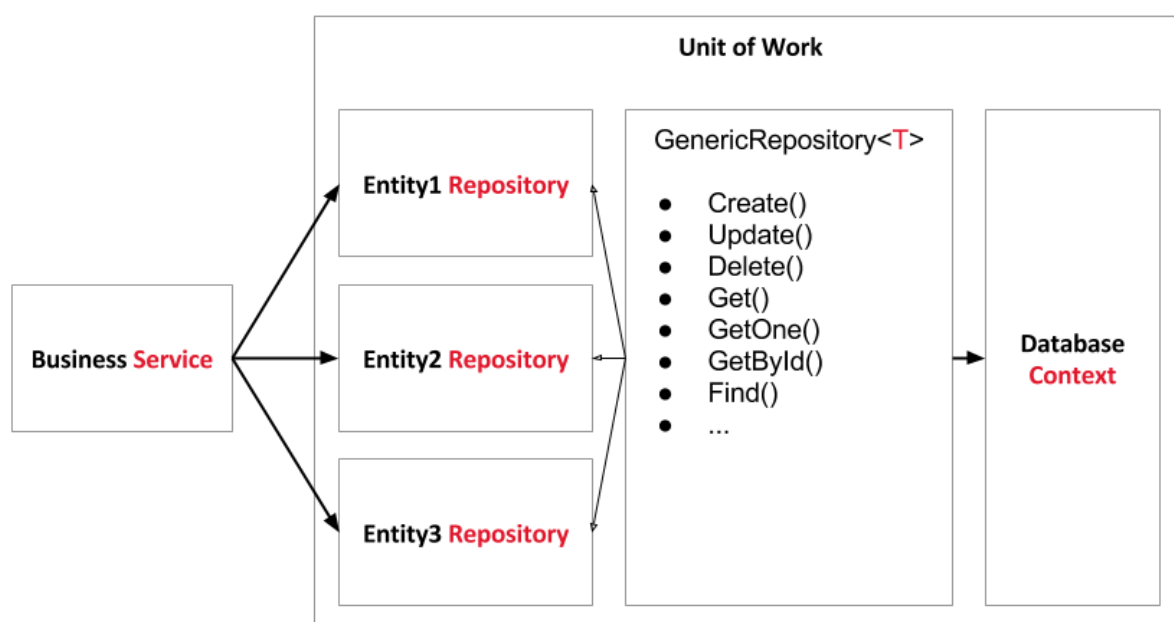
Actuellement les données permettant le calcul des recommandations sont stockées en mémoire vive. Ceci ne sera pas viable sur le long terme, nous prévoyons de passer vers un stockage rapide clusterisé (ex. Redis) ou vers un système dédié aux calculs d'algorithmes d'intelligence collective (ex. Apache Mahout).

## V. Designs patterns

Dans notre solution nous utilisons plusieurs designs patterns afin de faciliter l'accès à certaines parties du système, nous allons les voir en détail dans cette partie.

### Unit Of Work

Le design pattern Unit Of Work nous permet de travailler sur la base de données (DB) sans connaître l'implémentation (SQL Server, PostgreSQL, etc) de manière simplifiée.



*Schéma du pattern Unit of Work*

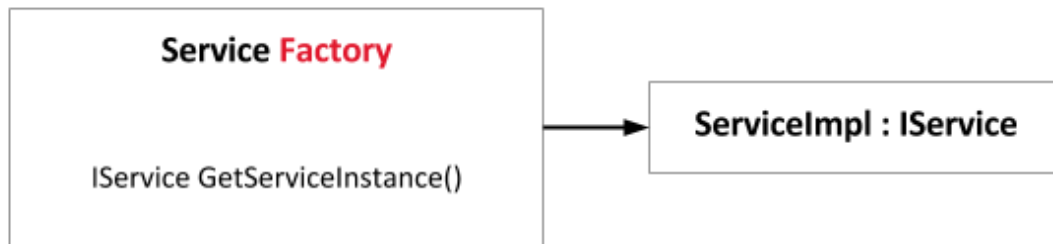
Comme le montre le schéma ci-dessus, nous travaillons avec une instance unique du contexte de base de données par Unit Of Work, ce qui permet l'unification du cache local avant l'application des modifications au SGBD (dans le cas de mewpipe, Microsoft SQL Server).

Nous proposons une repository par entité, qui représentera une table au niveau de la DB, Cette repository est une instance spécialisée de la classe générique `GenericRepository<T>` qui implémente des méthodes généralisées et simplifiées d'accès à la DB.

L'avantage principal pour nous est le partage d'un même "Database Context" dans une seule Unit Of Work afin de pouvoir travailler sur des tables à grand nombres de relations facilement.

## Factory

Notre architecture est basée sur des services que nous utilisons un peu partout afin d'effectuer des actions sur les vidéos. Ces services nous sont fournis par des “factories”.



*Schéma du pattern Unit of Work*

Le but de la factory est de séparer l'utilisation de la construction, en effet, instancier un service dans une classe peut être dangereux, si l'on change d'implémentation il faudra changer toutes les références au constructeur. C'est pourquoi nous utilisons une classe qui fait la construction pour nous, puis nous pouvons utiliser le service construit dans notre classe.

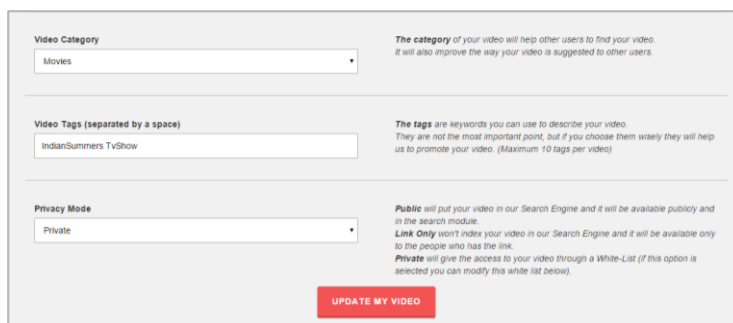
Notre factory, **VideoServiceFactory**, nous permet de récupérer des objets implémentant les interfaces **IVideoApiService** et **IVideoWorkerService**. Ces services sont respectivement les systèmes servant à l'API et au VideoWorker pour le traitement vidéo.

## VI. Ergonomie et expérience utilisateur

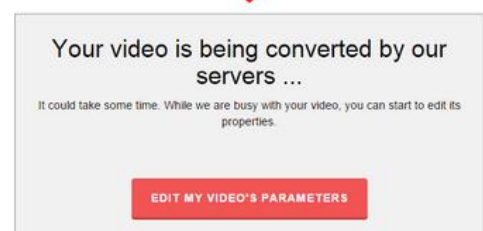
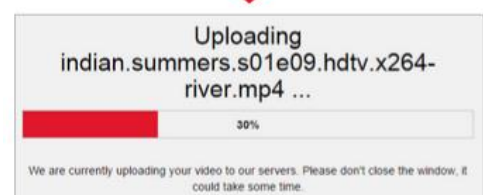
Durant le développement de MewPipe, nous avons choisi de mettre l'utilisateur au premier plan en développant la meilleure interface utilisateur possible. Nous allons maintenant mettre en avant certains points de l'expérience utilisateur que nous avons créée pour MewPipe. Cette liste est non - exhaustive, mais elle suffit à mettre en avant ce que nous souhaitons faire ressentir.

### Aide visuelle et dynamique pour l'utilisateur

Bien que l'utilisateur n'ait pas l'occasion de saisir souvent des données sur le site, l'ajout d'une vidéo requiert une interaction forte avec ce dernier, la première étant le transfert de la vidéo. Nous avons voulu proposer à l'utilisateur une manière simple et intuitive d'interagir avec la page. Un simple glisser déposer suffit à démarrer le transfert du fichier, dans le cas où il souhaite utiliser l'explorateur, il peut bien entendu y accéder via un simple clic.



The screenshot shows a form for uploading a video. It has three main sections: 'Video Category' with a dropdown menu set to 'Movies'; 'Video Tags (separated by a space)' with a text input containing 'IndianSummers TVShow'; and 'Privacy Mode' with a dropdown menu set to 'Private'. Each section has explanatory text. At the bottom right is a red button labeled 'UPDATE MY VIDEO'.



### Visuels des éléments d'interactions du site

Nous pouvons voir sur les visuels précédents que l'utilisateur est toujours tenu au courant de ce qui se passe et de pourquoi il attend. Il est également guidé lorsqu'il y a un champ à remplir. En effet nous avons conscience que le référencement d'une vidéo est important pour les éditeurs de contenu, nous souhaitons donc leur donner toutes les clefs pour optimiser la visibilité de leurs contenus.

## Édition rapide

Un utilisateur connecté peut vouloir vérifier que tout se passe bien sur sa vidéo en la regardant depuis la page de lecture, ou tout simplement recevoir un lien expliquant qu'il y a une erreur sur telle vidéo, il voudra alors modifier l'erreur. Nous avons choisi de proposer un lien d'édition directement sur la page de la vidéo, il retrouvera alors un lien "Edit Video" directement à droite du titre.

Indian Summers s01e09 [Edit video](#)  
Added the 12/06/2015 by Heavenstar

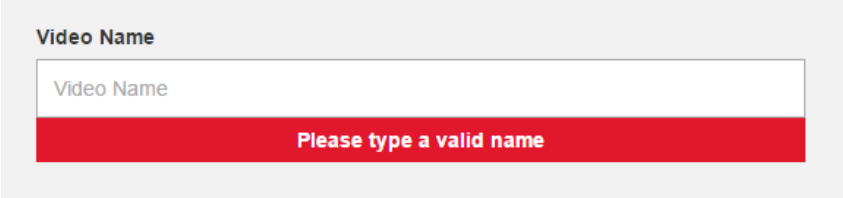
### *Visuel de l'édition rapide*

## Messages d'erreur pratiques

Afin de mieux guider l'utilisateur lors de l'utilisation du site, nous avons ajouté un système d'affichage d'erreurs constants. Nous avons deux types d'erreurs :

- Erreurs spécifiques aux formulaires
- Erreurs génériques

Les erreurs des formulaires sont affichées en direct dans le formulaire en question sous forme d'un bandeau rouge en dessous de la zone de saisie. Le visuel suivant montre le cas où l'utilisateur oublie de saisir un nom pour sa vidéo.

A screenshot of a web form for a video. At the top, it says "Video Name". Below this is a text input field containing the placeholder text "Video Name". Directly beneath the input field is a solid red horizontal bar with the white text "Please type a valid name" centered inside it.

### *Visuel des erreurs de formulaires*

Ce type d'erreur est constant sur tout le site, l'utilisateur peut donc clairement comprendre où il a fait une erreur.



Les erreurs génériques quant à elles, sont affichées différemment. Nous avons choisi de placer un bandeau très visible en haut de la page (à l'exception de la page de lecture de vidéos, où le bandeau est en bas de page afin de ne pas détériorer l'expérience de lecture de la vidéo).

Le bandeau est facile à faire disparaître, il suffit d'un simple clic sur ce dernier. Voici un visuel d'exemple :



### *Visuel des erreurs génériques*

#### Mise en avant de la vidéo

Quand l'utilisateur arrive sur la page d'accueil, la première chose qu'il voit est la liste des vidéos populaires du moment, cette liste prend la presque totalité de la page, car nous pensons que l'utilisateur vient ici pour voir des vidéos.

Une fois sur la page de lecture, la première chose que nous affichons est le lecteur, l'utilisateur est directement sur l'activité qu'il souhaitait faire, il peut ensuite choisir de descendre dans la page afin de voir le titre, la description, les recommandations, etc

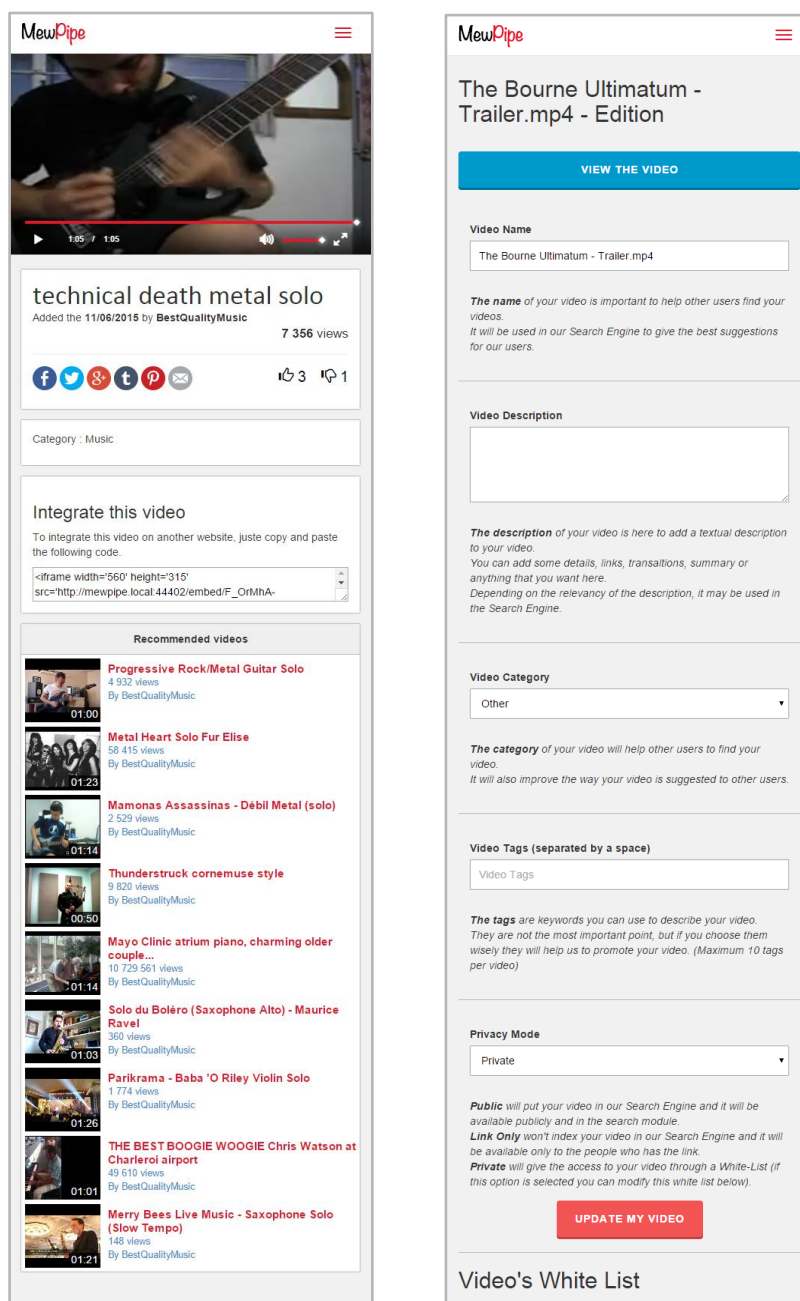


### *Visuel du lecteur vidéo*

## Site accessible sur mobile

Les habitudes de navigation sur internet changent, et le mobile prend une partie très importante sur le web. C'est pourquoi nous avons choisi de rendre compatible les mobiles avec MewPipe.

Tout le site s'adapte, les menus, le lecteur, les listes, les formulaires, etc. Les visuels suivants montrent la page de lecture et d'édition dans sa version mobile.



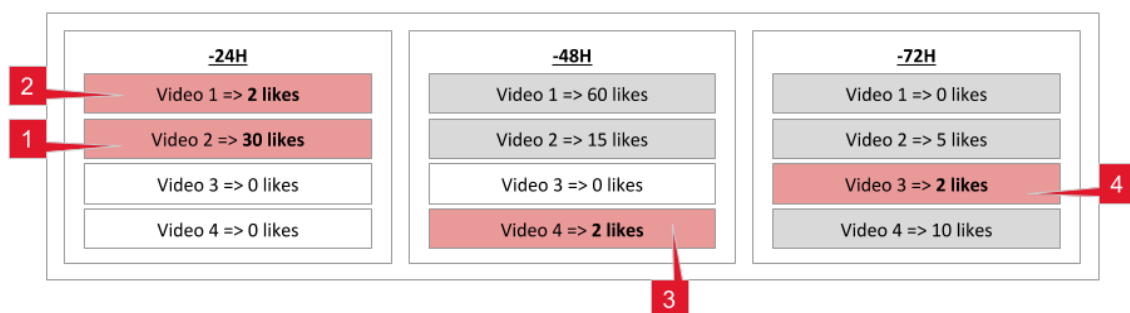
Visuel du mode mobile de MewPipe

## VII. Algorithmes

Nous utilisons deux algorithmes notables dans MewPipe, le premier étant celui qui permet de définir les vidéos affichées sur la page d'accueil, le second étant un algorithme d'intelligence collective qui permettra de déterminer les recommandations des vidéos. Nous allons étudier ces algorithmes en détail.

### “Tendances” de la page d'accueil

Les tendances sont définies à la volée, quand une demande est faite à l'API, le serveur va analyser les vidéos en fonction du nombre d'avis positifs (likes) dans les dernières 24h (ou celles antérieures s'il n'y a pas eu d'avis positifs récemment).



*Schéma de l'algorithme "Trending Videos"*

Le schéma ci-dessus nous explique la logique de fonctionnement de l'algorithme, son but est de trouver les vidéos appréciées dans le laps de temps le plus proche du temps actuel.

Dans le premier laps de temps (moins de 24h) nous avons deux vidéos disposant de likes, par conséquent la première vidéo du classement de la page d'accueil sera celle avec le plus de likes (la Vidéo 2) et la seconde (la Vidéo 1). Les deux autres sont ignorées car elles ne possèdent pas de données récentes.

Dans le second laps de temps (moins de 48h), deux vidéos sont ignorées car déjà ajoutées au classement, une troisième est également ignorée car elle n'a toujours pas de données et la dernière (la vidéo 4) dispose de 2 likes, par conséquent elle est ajoutée en 3ème position de la page d'accueil.

Le dernier laps de temps dispose de 3 vidéos sur 4 ignorées car déjà ajoutées au système et d'une vidéo disposant de 2 likes, la vidéo 3 est donc ajoutée en 4ème position de la page d'accueil.

Cette logique se suit jusqu'à ce que l'algorithme ait trouvé 40 vidéos, ce qui est assez rapide.

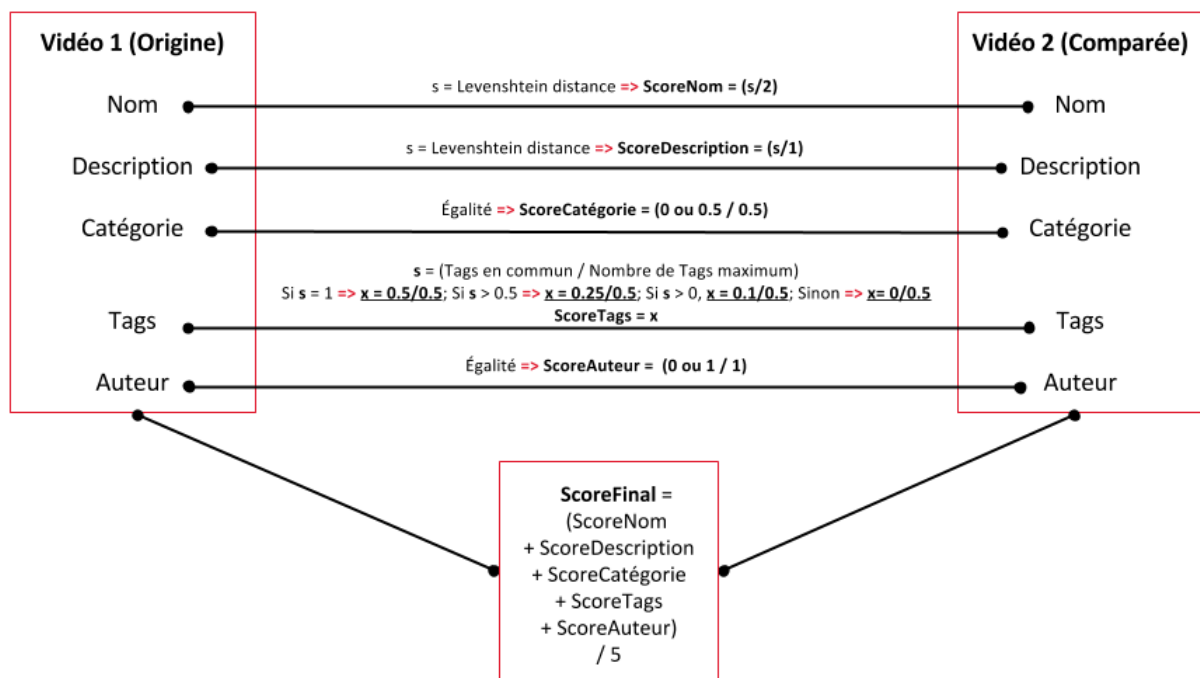
Il est à noter que cet algorithme est potentiellement améliorable pour faire ressortir des vidéos ayant une augmentation significative du nombre de likes, même si la vidéo est plus ancienne que d'autres.

## Recommandations

Notre algorithme de recommandations, est plus lourd et plus complexe que le précédent, c'est pourquoi il n'est pas exécuté à la volée. En effet, nous avons un **RecommenderEngine**, une application qui tourne sur une machine dédiée, qui va s'occuper de calculer ces recommandations en boucle afin d'avoir les données les plus à jour possible.

Nous nous basons sur un score de comparaison entre deux vidéos afin de récupérer les meilleures vidéos. Nous avons mis en place deux manières de calculer ce score. La première est basée sur les données textuelles (titre, description, tags) et les données contextuelles (catégorie, utilisateur) qui accompagnent la vidéo. La deuxième est, quant à elle, basée sur les données sociales (les appréciations utilisateurs).

### Analyse textuelle et contextuelle



### Schéma de l'algorithme de recommandations basées sur le contenu/contexte

Comme le montre le schéma précédent, nous utilisons plusieurs formules de calcul pour déterminer un score qui nous permet de comparer deux vidéos.

Le score final est donc déterminé par plusieurs critères, la catégorie et l'auteur sont les deux critères les plus simples, ils se contentent de comparer l'égalité pour déduire une valeur.

Le nom et la description utilise la Distance de Levenshtein ([http://fr.wikipedia.org/wiki/Distance\\_de\\_Levenshtein](http://fr.wikipedia.org/wiki/Distance_de_Levenshtein)), pour résumer simplement, cette méthode nous permet de calculer la similitude entre deux chaînes de caractères, nous l'utilisons pour savoir si deux chaînes se ressemblent (par exemple dans le cas de deux épisodes d'une série, où seul le numéro de saison/épisode risque de changer).

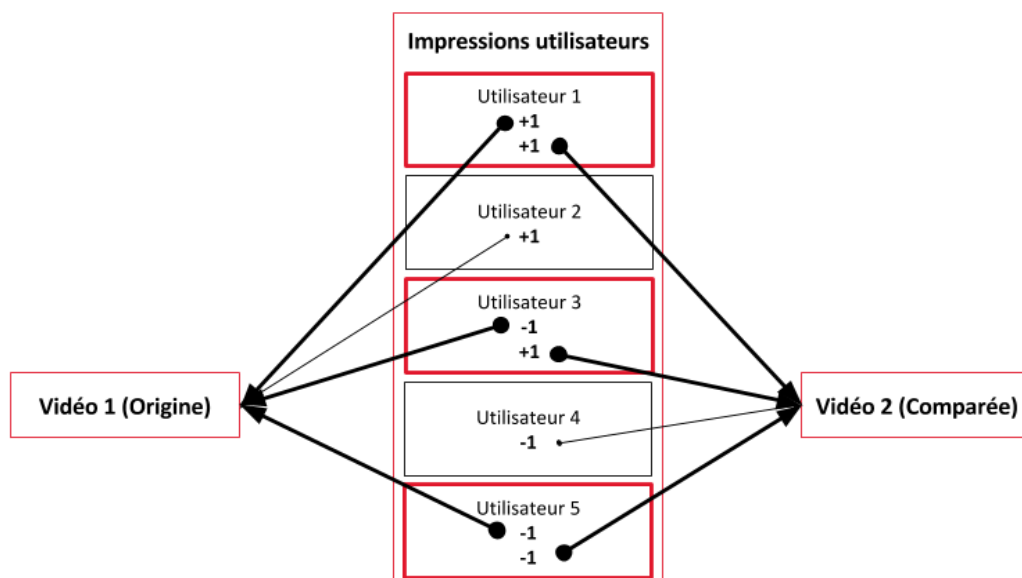
Enfin, les tags sont comparés par un pourcentage de similarité, le calcul est expliqué dans le schéma.

Ce score final nous permet de comparer toutes les vidéos à la vidéo en cours pour déterminer les vidéos les plus proches de cette dernière.

Nous avons pu voir ce premier système, nous nous en servons surtout pour les vidéos qui viennent d'être ajoutées à MewPipe et qui n'ont pas encore d'impressions des utilisateurs.

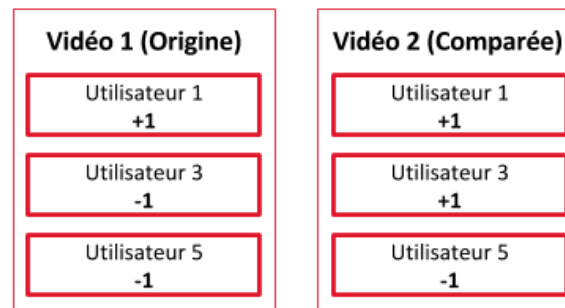
### Analyse des impressions utilisateurs

Ce second système nous permet de proposer aux utilisateurs des données basées sur l'intelligence collective (données collectées par MewPipe) afin de leur proposer des vidéos grâce aux préférences de tous les utilisateurs de MewPipe.



*Schéma de l'algorithme de recommandations basées sur les impressions*

Le schéma précédent nous montre le début de ce système, ici nous récupérons les impressions utilisateur dont nous avons besoin pour déterminer le score d'une vidéo par rapport à une autre. La technique est simple, nous voulons les impressions des utilisateurs qui ont donné une impression pour la vidéo d'origine ET pour la vidéo comparée, ainsi nous pouvons déterminer quelles vidéos les utilisateurs ayant apprécié cette vidéo ont aussi regardés. Les impressions peuvent être négatives ou positives.



*Schéma des collections pour les vidéos 1 et 2*

Une fois cette sélection faite, nous nous retrouvons avec une collection d'appréciations pour la Vidéo 1 et une collection pour la Vidéo 2 (les appréciations vont par paires entre les deux collections, en effet, nous ne regroupons que des appréciations d'utilisateurs ayant donné leur avis sur la Vidéo 1 et la Vidéo 2).

Afin de déterminer un score pour la vidéo, nous utilisons la méthode du coefficient de corrélation de Pearson ([https://en.wikipedia.org/wiki/Pearson\\_product-moment\\_correlation\\_coefficient](https://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient)) qui va nous permettre de déterminer une valeur de corrélation pour ces deux vidéos, basée sur les impressions utilisateurs.

### Traitement des résultats des deux systèmes

Une fois que les deux systèmes précédents ont été appliqués, nous nous retrouvons avec une collection de correspondances Score => Vidéo pour le premier système et une deuxième collection identique (mais avec des résultats différents) pour le second système.

Afin de récupérer la liste de recommandations finales pour la vidéo en cours, nous allons tout simplement créer une nouvelle collection en insérant la vidéo ayant le plus grand score (entre les deux collections précédentes) dans la collection. Nous ferons ceci 20 fois afin d'avoir un nombre suffisant de recommandations.

## VIII. Points particuliers

### Files d'attentes

Afin d'éviter d'encombrer notre système, nous avons choisi de déléguer les conversions vidéos à des "Video Workers" qui feront le travail en arrière-plan. Cependant il faut notifier ces workers afin qu'ils puissent convertir la vidéo dès que cette dernière est prête. Pour faire cette communication interne, nous avons choisi d'utiliser une file d'attente.

Une file d'attente, en informatique, est un composant utilisé pour transmettre des messages entre des processus ou des parties d'un service. Nous avons choisi d'utiliser le logiciel RabbitMQ (<https://www.rabbitmq.com/>) qui est lui-même basé sur le protocole AMQP (<https://www.amqp.org/>). AMQP est un standard dans l'informatique, ce qui implique qu'il est facile d'ajouter d'autres composants sur la solution.

Notre scénario d'utilisation est le suivant : un utilisateur transfère une vidéo au VideoRepository (le site qui gère les transferts de fichiers), une fois le transfert terminé, le VideoRepository envoie un message via la file d'attente.

La file d'attente va ensuite délivrer les messages dans l'ordre d'arrivée à chaque video worker disponible.

Si le video worker fait une erreur et crash, le message retourne en première place dans la file d'attente pour être redistribué au prochain video worker disponible.



*Schéma de fonctionnement d'une file d'attente*

Le but principal de cette implémentation est d'éviter un engorgement d'une des parties du système.

## Gestion des fichiers vidéo

La gestion des fichiers vidéos sur MewPipe peut être vue sous la forme d'un pipeline.



*Schéma du traitement vidéo sur MewPipe*

Nous avons déjà détaillé plusieurs parties de ce traitement, mais ici nous pouvons voir la suite logique des évènements.

**La première étape** consiste tout simplement au transfert du fichier par l'utilisateur sur le serveur.

**La seconde étape** est d'envoyer un message pour informer les workers qu'une nouvelle vidéo est prête à être convertie.

**La troisième étape** est celle que nous allons détailler ici. Le traitement de la vidéo implique plusieurs étapes.

- Récupération du fichier original
- Analyse des méta-données du fichier original
- Récupération d'une image miniature de la vidéo (à environ 25% de la vidéo)
- Calcul de la résolution maximum de la vidéo (360p, 480p, 720p ou 1080p)
- Conversion de la vidéo dans les formats MP4 et OGG dans chacune des résolutions possibles (si la résolution maximum déterminée est 720p, la vidéo sera convertie en 360p, 480p et 720p)
- Stockage en base de données de ces fichiers convertis
- Suppression du fichier original en base de données
- Ajout des méta-données de la vidéo en base de données (durée de la vidéo par exemple)
- Envoi de l'email de fin de conversion à l'utilisateur
- Mise à jour du statut de la vidéo en "Publiée" dans la base de données

Le nombre d'étapes est assez important, mais chacune de ces étapes permet d'assurer une vidéo de qualité correcte, une bonne compatibilité avec les différents navigateurs web et périphériques ainsi que d'assurer une extensibilité optimale.



Nous utilisons **FFmpeg** (<https://www.ffmpeg.org/>) via la bibliothèque **NRECO.VideoConverter** ([http://www.nreco.site.com/video\\_convert\\_net.aspx](http://www.nreco.site.com/video_convert_net.aspx)) pour convertir les fichiers vidéos.

**La quatrième étape** consiste tout simplement à lancer l'algorithme de recommandations afin de récupérer des recommandations basées sur les données textuelles et contextuelles.

## OAuth 2.0

Nous utilisons le protocole OAuth 2.0 pour MewPipe afin d'ouvrir l'accès à l'API aux développeurs. En effet, grâce à ce système, nous pouvons fournir un couple de clefs aux développeurs pour qu'ils puissent intégrer notre système dans leurs applications.

Nous n'allons pas détailler le fonctionnement du protocole dans cette documentation, cependant vous pouvez en trouver tous les détails ici : <https://tools.ietf.org/html/rfc6749>.

L'implémentation de notre serveur pour OAuth 2.0 est réalisée entièrement par nos soins en suivant la RFC.