TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Mikkel Paat 213223IADB

Online Video Archival System

Building Distributed Systems individual project

Supervisor: Andres Käver
MSc

Tallinn 2023

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Mikkel Paat 213223IADB

INTERNETIVIDEOTE ARHIVEERIMISE SÜSTEEM

Juhendaja: Andres Käver
MSc

Tallinn 2023

Author’s Declaration of Originality

I hereby certify that I am the sole author of this document. All the used materials, references to the literature and the work of others have been referred to. This document has not been presented for examination anywhere else.

Author: Mikkel Paat

06.06.2023

Abstract

This document is a project proposal for the individual home project in the subjects "Building Distributed Systems", "ASP.NET Web Applications" and "JavaScript".

The document starts with an introduction, which describes the problem that the project is meant to solve and outlines rough expectations for the end result. The main part of the document is an analysis of various aspects of the project and how they should work. This is accompanied by an ERD (Entity Relationship Diagram), which can be found in Appendix 2 and is designed by taking into account the considerations outlined in the analysis. After that, the document also provides some sketches and thoughts on the potential UI (User Interface) design and flow of the final web application. Finally, the document also features a retrospective chapter written at the end of the project, which describes the end result and some of the more noteworthy aspects of the process of its creation.

The document is written in English and contains text on 22 pages, including 4 chapters, 8 figures and 0 tables.

Annotatsioon

Internetivideote arhiveerimise süsteem

See dokument on õppeainete "Hajussüsteemide ehitamine", "ASP.NET veebirakendused" ja "JavaScript" koduse individuaalprojekti püstitus.

Dokument algab sissejuhatusega, mis kirjeldab probleemi, mida projekt lahendama peaks ning toob välja umbkaudsed nõuded lõpptulemuse jaoks. Dokumendi peamine osa analüüsib loodava süsteemi erinevaid aspekte ning kuidas need töötada võiksid. Sellega käib kokku teises lisas leiduv olemi-suhtediagramm, mis kirjeldab süsteemi andmebaasimudelit, mis on disainitud analüüsis väljatoodud kaalutlusi arvestades. Seejärel on dokumendis planeeritava veebirakenduse kasutajaliidese kohta mõned joonised ja kirjeldused. Viimane peatükk on projekti lõpus kirjutatud tagasivaade, mis kirjeldab lõpptulemust ja selle loomisprotsessi olulisemaid aspekte.

Dokument on kirjutatud inglise keeles ning sisaldab teksti 22 leheküljel, 4 peatükki, 8 joonist, 0 tabelit.

List of Abbreviations and Terms

Platform	A website on which videos may exist
Author	A user on a platform
Entity	General term for several concepts being modelled or dealt with in the system being designed
API	Application Programming Interface
Webhook	A callback function that allows lightweight, event-driven communication between 2 APIs[1]
Pub/Sub	A specific kind of messaging service that supports webhooks, among other things[2]
ERD	Entity Relationship Diagram
UI	User Interface
JSON	JavaScript Object Notation
DTO	Data Transfer Object
DAL	Data Access Layer
HTML	HyperText Markup Language
CORS	Cross-Origin Resource Sharing

Table of Contents

List of Figures

1. Introduction

Online video consumption has become an important part of many people's lives in the modern day. With people spending so much of their time watching all sorts of videos, plenty of those videos will likely end up being important and memorable experiences for their viewers. Whether they provide entertainment, educational value or cultural insight, online videos are a form of media, art and even history that deserve to be preserved.

Currently, however, people tend to simply watch whichever videos are fed to them by various websites' algorithms, and then move on. There is no guarantee that these experiences will still exist in a few months or years or decades. The video is still hosted on a platform out of the viewer's control, and might become inaccessible at any time because the author changed the video to private, a copyright holder issued a copyright strike, or perhaps even the entire video platform was shut down.

While there exist some ways to download videos from various online video platforms, they all have their shortcomings and they don't provide any sufficient way to manage the vast amounts of video content that a person might wish to archive over the course of their life. A video platform might not even provide a download feature, or such a feature might be locked behind a paywall or limit what the user can do with the downloaded video. There is usually no provided way to archive the context surrounding the video, such as the comments, or the history of different changes that might have been made to the video or its metadata.

The goal of this project is to create a system that allows its users to conveniently archive videos from supported video platforms. Rather than being one large centralized service hosted by one owner and used by large amounts of users, it's meant to be a project that people can self-host with the means available to them, potentially sharing their instance with a small group of people. Initially the only supported platform will be YouTube, but if time allows, Twitch support may also be added. The system must support both manually archiving specific videos and automatically archiving videos from certain authors or playlists. Archiving a video involves archiving its surrounding context, such as the video's title, description and comments. Usually the video file is also downloaded, though that can optionally be turned off. In addition to videos, the system will also support archiving playlists and authors. When a video, playlist, or author is archived, it will usually also be monitored - this means that the system will keep track of changes made to it and automatically record those as well.

The second major aspect of the system will be content management and organization. Users will be able to define and assign categories to videos, authors and playlists. They will also be able to give detailed ratings to videos, authors and playlists - in the context of a specific category or not. This will allow users to filter and catalogue their archived videos, video series, subscriptions etc. according to various conditions, as opposed to simply having a pool of content and data without much meaning attached to any of it.

In addition to and using the aforementioned features, the system will provide some enhancements on top of existing platforms. For example, the system will allow users to subscribe to specific playlists. That way, if a user is only interested in specific content from an author, they can subscribe to the playlists that interest them, without being flooded with notifications about other videos. Another enhancement stems from the system's category system, which will enable users to filter their subscriptions by category and see the ratings they've given these authors in the past. If a user is subscribed to hundreds or even thousands of channels, half of which the user doesn't even remember subscribing to, this feature will significantly improve the user's ability to keep track of all these authors, whose content the user likely wishes to keep up to date with. Users will also be able to create playlists that contain videos from several different platforms.

The system will consist of a backend and several frontends. The backend will be written in the C# programming language and contain a library for interfacing with the system's database, which can be used by other parts of the system. The backend will also contain automated monitoring and archival services, which will run in the background and regularly perform tasks necessary for automatic archival and notifications, such as checking for new video uploads or changes in monitored data. There will also be an ASP.NET Core web application that will function as a web-based client for the whole system, but also provide an API interface which will allow for the creation of other client applications. One of these other client applications will be a JavaScript-based web application, which should have full or nearly full feature parity with the ASP.NET Core web application. There are also plans to create a browser extension and a Discord bot to provide some more convenient ways to quickly perform some simple functions with the system, but these are not within the scope of this specific project.

2. Analysis and planning

When specific attributes or entities are mentioned in this chapter, Appendix 2 should be referred to in order to get a better understanding of said attributes or entities.

2.1 Platforms and subplatforms

A platform is any specific location that videos and entities related to videos belong to. One of these platforms is this archival system, but all the rest are external websites such as YouTube or Twitch. However, videos from the same website could belong to a different sub-section of that website. For example, videos from YouTube could be divided into regular videos, shorts, clips, and livestream recordings. Videos from Twitch could be divided similarly. Depending on the platform, there could be a significant difference between how these different types of videos should be treated – they could have different URL patterns, different properties, and even different API endpoints. For this reason, such videos will be treated as being from different platforms. However, in some contexts it may still be desirable to treat all these videos as being from one platform, so such platforms should be defined as subplatforms of a common parent platform.

2.2 Authors and users

This project will use the ASP.NET Core Identity framework to facilitate users, permissions and logging in. A user in this context is semantically different from a user of a video platform, who can upload videos, leave comments, create playlists etc. This becomes apparent as soon as one attempts to define a database entity that would store both of these kinds of "users" – there would be one group of attributes that only pertain to one type of user, and another group of attributes that are only relevant to the other type of user. Thus, it makes sense to separate the two concepts – henceforth the kind of user entity that can log into the archival system will be called a user, and the kind of user entity that is intrinsically semantically tied to video content will be called an author.

However, users of the archival system still need to be able to behave as an author – for example when creating a playlist in the archive. In fact, users should have the ability to act as several authors (in the archival system), which could be used as a means to present themselves with different profiles in different contexts or to further segment and organize their archive library under different identities.

2.3 Entity attributes and nullability

Most of the main archive database entities, such as videos, authors, playlists, and comments have many attributes which are nullable – they might not have a value assigned to them. This is usually due to the following reasons:

- A property that exists on one platform may not exist on another. For example, perhaps some platforms' videos don't have descriptions.
- The value might not be known to the archival system. This may be because the entity itself wasn't accessible – the author's account was deleted or the video was set to private – or because that attribute isn't publicly accessible, like the dislike count on YouTube since December 13, 2021[3].
- The value might actually be empty. For example, a specific video might not have a default audio language set, even if videos on that video's platform usually do.

There are, however, some attributes which aren't nullable. The potential reasons for this are:

- That value is a meta-property assigned by the archival system and is always known. For example the Monitor attribute of the Video entity, which indicates whether the archival system should monitor changes to that video or not.
- That value is always necessary for the entity to even have any meaning in the archival system. A good example of this is the IdOnPlatform attribute, which stores a value that uniquely identifies that entity on its platform. Without that value the system wouldn't even know if that entity has already been added to the archive, and consequently allow for unlimited duplicate entries to be added.

2.4 Subscriptions and polling

Authors can subscribe to other authors. This may mean an author on the archival system subscribing to another author, who can be from any platform. Alternatively, this could mean an author on a platform subscribing to another author on that same platform. In both cases, the archival system will need to monitor the author who was subscribed to in order to be able to notify the subscriber of that author's video uploads. The system must also be able to monitor some other entities, such as videos, playlists and comments in order to update them in the archive if any changes are made to them on their respective platforms.

To accomplish this, the system will use one or both of the following approaches:

- Polling. This method involves periodically fetching data about the relevant entity from its platform to see if any changes have been made. The data could be fetched using API requests or using web scraping, depending on the specific platform and entity.
- Pub/Sub or Webhook. With this method the archival system will subscribe to a notification feed on the relevant external platform, which will notify the archival system if updates are made to the subject of the subscription. The upside of this is that changes will be received right around the time when they took place, as opposed to whenever the scheduled polling service gets around to checking for changes.

  However, this method may not be available on some platforms and even if it is available, is usually limited in the sorts of updates it supports. For example, the YouTube API's Pub/Sub only sends notifications when a video is uploaded or when a video's title or description are updated.[4] Since both YouTube's and Twitch's support for this method is author-specific, the archival system's design will also only support storing information about author-specific Pub/Sub subscriptions.[5] Thanks to this, the system won't need to store the subscription feed URL, because it can be constructed from the associated author entity.

  Regardless of the specific platform-specific nuances, although this method can be used to receive faster updates for some properties of some entities, use of polling will still be required to cover other entities and attributes and to ensure changes aren't missed in case of a notification not being received despite a relevant change taking place.

In order to receive notifications within the archival system according to a subscription between authors on an external platform, a user of the archival system must be able to give the archival system permission to access their subscriptions on that external platform. To accomplish this and to continuously synchronize these subscriptions, the system will use the ExternalUserToken entity. In addition to storing the necessary information for getting subscription data from the relevant external platform, this entity will also provide a way of linking users to these external authors, in order to determine which users should receive notifications about those subscriptions.

2.5 History

When certain types of entities are changed, the archival system shouldn't just discard the previous state of that entity. How a video's title or an author's profile picture has changed over time is an important aspect of these entities' identities. Additionally, some attributes, such as a video's captions, might be an essential accompanying element of the video. In extreme cases, even the video content itself might change. Or, as a less extreme case, a playlist's composition will almost definitely change as time flows.

All these kinds of changes must be saved into the archive. For each of authors, videos, playlists, and comments, a separate history entity will be used, containing a copy of many of the attributes in its respective original entity. Of course, some attributes will not be stored in the history, such as meta-attributes used by the archival system for monitoring, or attributes which should never change, such as an entity's platform.

Unfortunately this approach does mean that the data that didn't change will be duplicated in the history entities. However, that should be outweighed by the benefits of using such history entities, like the relative simplicity that is provided by treating them as snapshots of an entity's state that contains most of the relevant data inside it. The entities' relationships to other entities still remain with the main, "non-history" entity, which also simplifies matters by not requiring complicated join statements, which might potentially be caused by alternative approaches. And since a history entity will almost always be viewed in the context of its corresponding current entity, the attributes and relationships that aren't stored in the history entity can be accessed without any issues. Furthermore, it's expected that updates to such entities will be a somewhat infrequent occurrence, which lessens the impact of unchanged data being duplicated in these entities yet again.

If an entity's privacy status or availability changes, then that change will be tracked in a separate dedicated StatusChangeEvent entity. This is because an entity's privacy or availability changing is, in its nature, a different event from that entity's other attributes changing. Tracking these events separately makes it easy to create a separate timeline of video status changes and provides an entity to reference when notifying users who added that entity of those status changes. Also, such a status change event is unlikely to occur in conjunction with another attribute changing, so this reduces unnecessary data duplication in history entities.

2.6 Queue

The host of the archive system might wish to allow some users to suggest adding new entities to the archive, but not actually add them. Other users might be allowed to add an entity to the archive, but not to schedule it and its contents to be automatically downloaded. These kinds of archival requests should be stored in a queue until they're approved and potentially modified by a user with the authority to do so. This queue will also allow the client that added an item to the queue to store extra information alongside that item. When certain events happen to the queued item or the video, playlist, or author related to it, that extra information can be used to send a notification to the client that added it. For example, a Discord bot could use it to store information about which Discord user added a video, in order to later notify said user about that video's deletion.

2.7 Services

Many of the system's tasks need to run in the background, without direct user interaction. These should be implemented as separate services that perform their tasks independently. For example:

- A service for monitoring changes to entities and recording them in the archive.
- A service for refreshing external user tokens
- A service for sending notifications to users and authors

2.8 Categories, subcategories and ratings

Categorizing archived content properly is essential if the archived data is to maintain any of its meaning. Most categories will likely be defined by the archival system's users for themselves, but the system will also archive categories from other platforms. Unlike many other platforms, the archival system will allow its authors to assign categories to authors and playlists in addition to videos. Such category assignments should also specify whether or not sub-entities of the assigned entity should inherit its category – if an author who is in a category uploads a video, should that video also be assigned that same category? The system should probably also support sub-categories, for situations where a category only really has a meaning in the context of another.

This sort of extensive category system will allow users to effectively organize and filter videos, authors, and playlists they're interested in.

Another feature for organizing and providing meaning to archived content will be ratings. It's fairly common to allow users of a video platform to rate videos – even if it is with only a down- or upvote. Similarly to categories, the archival system should extend this functionality. Ratings will be applicable to playlists and authors as well. They will use a 5-star rating system with half-star precision. Most importantly, authors will be able to leave category-specific ratings: they might give a playlist a 5-star rating in the "Comedy" category, but an overall 1.5-star rating with no category specified. This will let authors better remember what a certain content creator or piece of content meant to them in the past or even who they were in the first place. To help with this, the ratings should also have an optional comment field.

2.8.1 Games

Games are enough of a big subcategory that they deserve to be recorded separately. If archiving content from a platform like Twitch, where almost every piece of content has a game attached to it, that association must also be reflected in the archive. The archival system will use IGDB[1] as a provider for information about existing games, but it could also allow users to add their own custom games for each video. Also, just in case, the system should support the possibility of a game only being associated with a video for a certain part of that video.

2.9 Notifications

An advanced notifications system is an integral part of the value that the archival system will provide to its users. Rather than having one single notifications feed, users must have the option of filtering their notifications in several ways. Filtering by different categories is essential and if a user has multiple authors related to it, the user should be able to also filter notifications using them.
Finally, when subscribing to an author or playlist, the subscribing author should be able to assign a numeric priority to that subscription, which will be added to notifications caused by that subscription and can then be used to filter and sort notifications according to their priority.

Notifications should also be sent to the user who requested the addition of an entity into the queue, if that entity is deleted or becomes unavailable for some reason.

2.10 Custom objects and localizations

Some attributes, such as a video's title and description, can have multiple versions for different languages and locales. Others, such as information about a video's local files, would need to be a list of custom objects with their own attributes. Since such data can't be directly stored in a database column, the typical approach would be to create a separate database entity for each of them and define a relation between them and the entity they belong to. However, such an approach would require complicated join queries when fetching or operating with the data and might also complicate the situation with history entities. Thus, for the sake of simplicity and because these attributes don't need to have relations with any other entities, these attributes will be serialized using JSON (JavaScript Object Notation) and stored in one database column each.

[1] IGDB link: https://www.igdb.com/about

3. Web Application UI sketches

This chapter provides rough sketches of some parts of the web applications' pages and page elements, to give a more concrete idea of what should be expected from the final project. However, it should be noted that these are only rough incomplete prototypes showcasing a limited subset of the web application's functionality and will thus likely change significantly over the course of development.

Common to every page in the web application will be a navigation bar at the top of the page. Figure 1 shows a section of the navigation bar with a text input field for quickly adding a video, playlist or author to the archive. There is also a bell icon for accessing the logged in user's notifications, an element for the user to select which of their author identities the user is currently acting as, and finally an element indicating who the logged in user is.

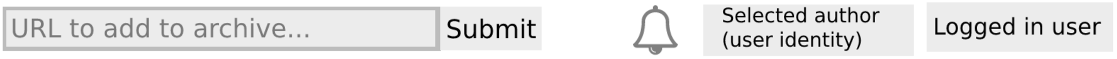


Figure 1. *An image of part of the navigation bar at the top of the web application.*

If the notification bell icon is clicked, the user will see a notifications view. As shown in Figure 2, the notifications view consists of tabs for viewing different kinds of notification feeds, options for filtering the selected notification feed, and the notification feed itself. The user can mark notifications as read or unread and click on a notification to be taken to a relevant page in the web application. The notifications view may end up being a separate web page or it may be integrated into the navigation bar, to pop up upon clicking on the notification bell. This (and other such features) might differ between the JavaScript-based web application and the ASP.NET Core web application, due to some features potentially being difficult or impossible to implement without JavaScript.

Upon clicking on a video upload notification, the user will be taken to the video viewing page. Figure 3 shows what the video viewing page might look like. It should be noted that the view displayed in Figure 3 is showing the video in the context of a playlist that contains the video. This view could be arrived at by clicking on a notification from a playlist subscription or by clicking on a video within a playlist.

Overall, this page is quite similar to the layout of a YouTube video's watch page, containing a video player in the center, with some action buttons and information about the video

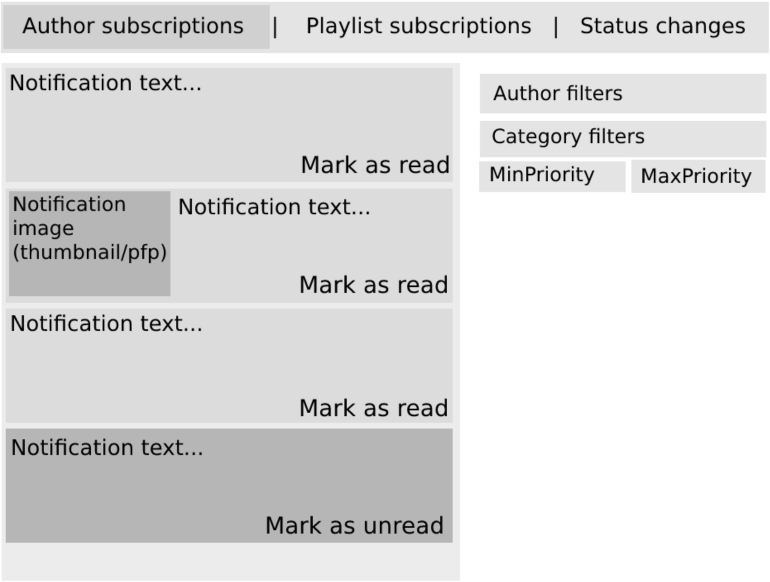


Figure 2. *An image of the notifications view in the web application.*

and its author below it and comments even further below those. However, there are some additional elements, such as the video's external privacy/availability status, timestamps indicating when the video's information was last fetched from its native platform. The subscribe button has been augmented with a priority option, which will allow the user to set a numeric priority to their subscription. There are also buttons for downloading the video from the archive, viewing the video's change history, selecting whether the video should be played from the archived video file or directly from its website of origin using an embedded video player etc. As was the case with the notifications button, some of these buttons might end up leading to dedicated pages for performing their actions, whereas others could simply function as pop-up menus within the video page. Again, this will likely differ between the JavaScript-based web application and the ASP.NET Core based web application.

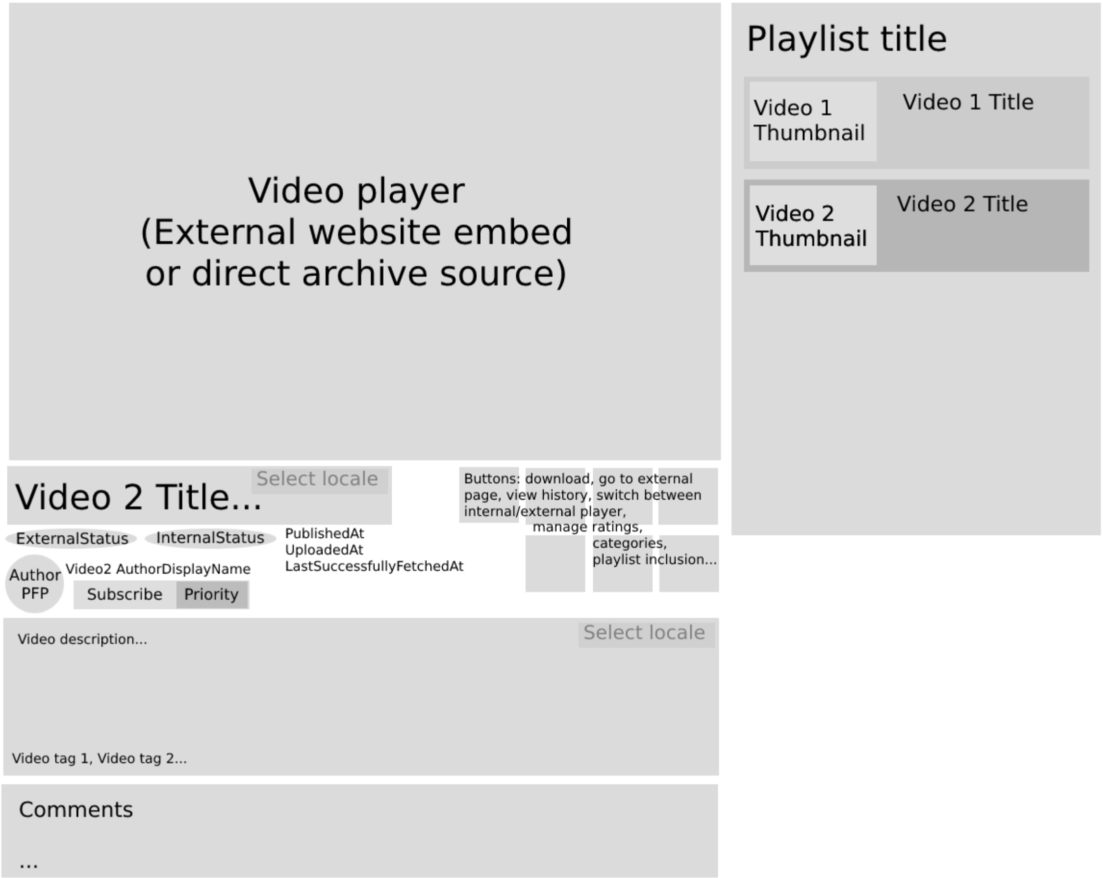


Figure 3. *An image of the main video viewing page in the web application.*

The image in Figure 4 demonstrates what the menu for adding ratings to an entity might look like. Important to note is that it allows for one entity have multiple ratings with each being in the context of a category (although it should also allow adding one rating without a category). There is also an optional comment field. Yet again, this will likely end up being a separate web page in the ASP.NET Core version of the web application and a pop-up menu in the JavaScript-based version.

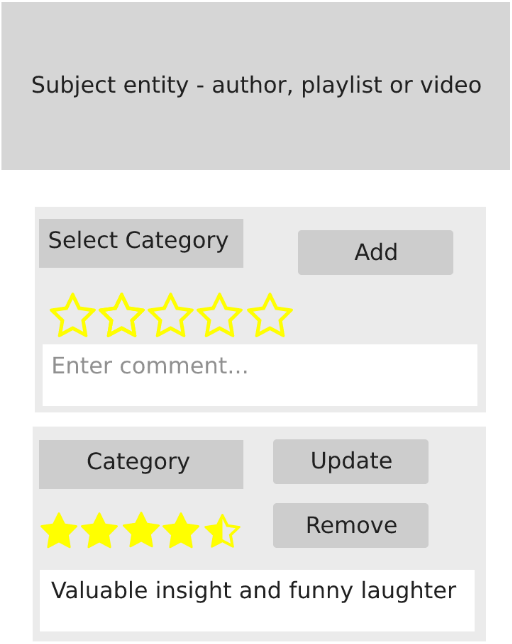

Subject entity - author, playlist or video
Select Category
Add
Enter comment...
Category
Update
Remove
Valuable insight and funny laughter

Figure 4. *An image of the view used for managing an entity's ratings in the web application.*

Figure 5 features an author page, with its videos section selected. Most of it should be fairly intuitively understandable, however it should be noted that some elements, such as the ones labelled "Category filtering options" and "Sorting options" are simplified placeholders – their implementation details will be determined at a later point and they will probably end up looking somewhat more complex that simple rectangles in the completed web application.

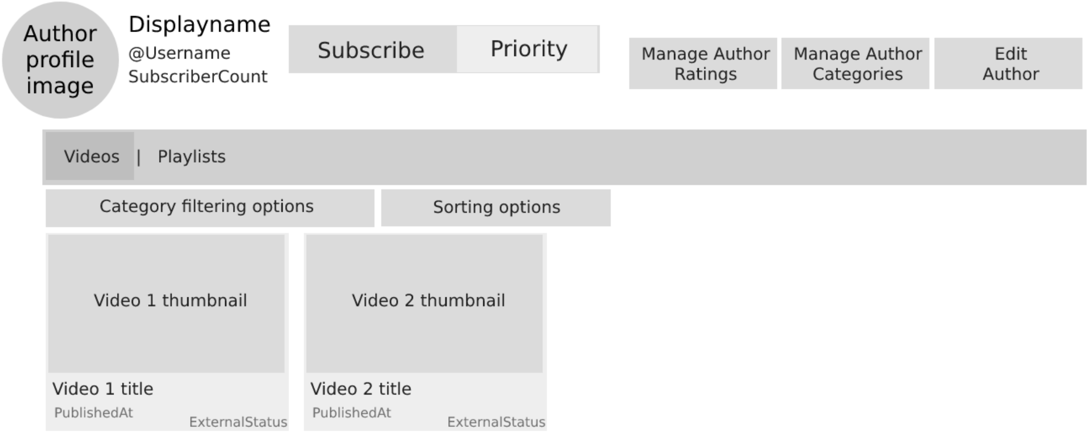


Figure 5. *An image of the videos section of an author's page in the web application.*

In Figure 6 an editable playlist page can be seen. Of note is the way that videos can be moved around in the playlist using a numeric text input field and a submit button for each video. This approach is necessary for the JavaScript-less web application, since a drag-and-drop rearranging mechanism can't be easily implemented without JavaScript. The "Sorting options" element above the list of videos also deserves attention – if a non-manual ordering method is selected, the user shouldn't be able to move videos around anymore, because the ordering will be automatic and thus not customizable.

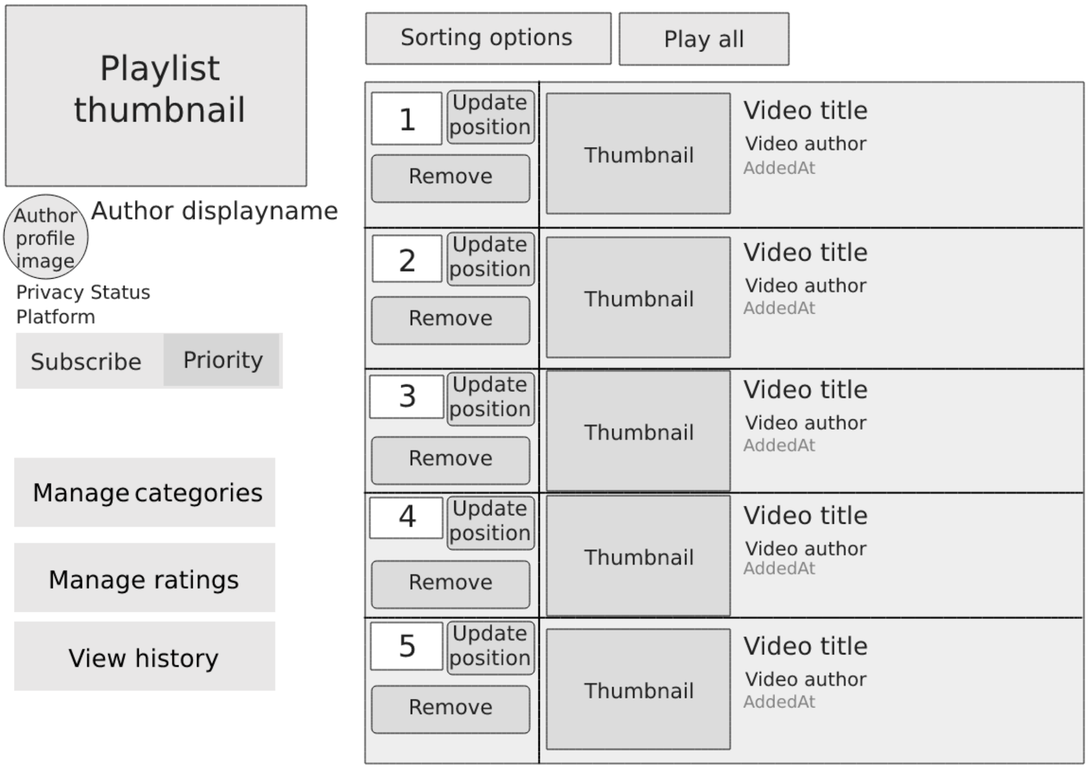
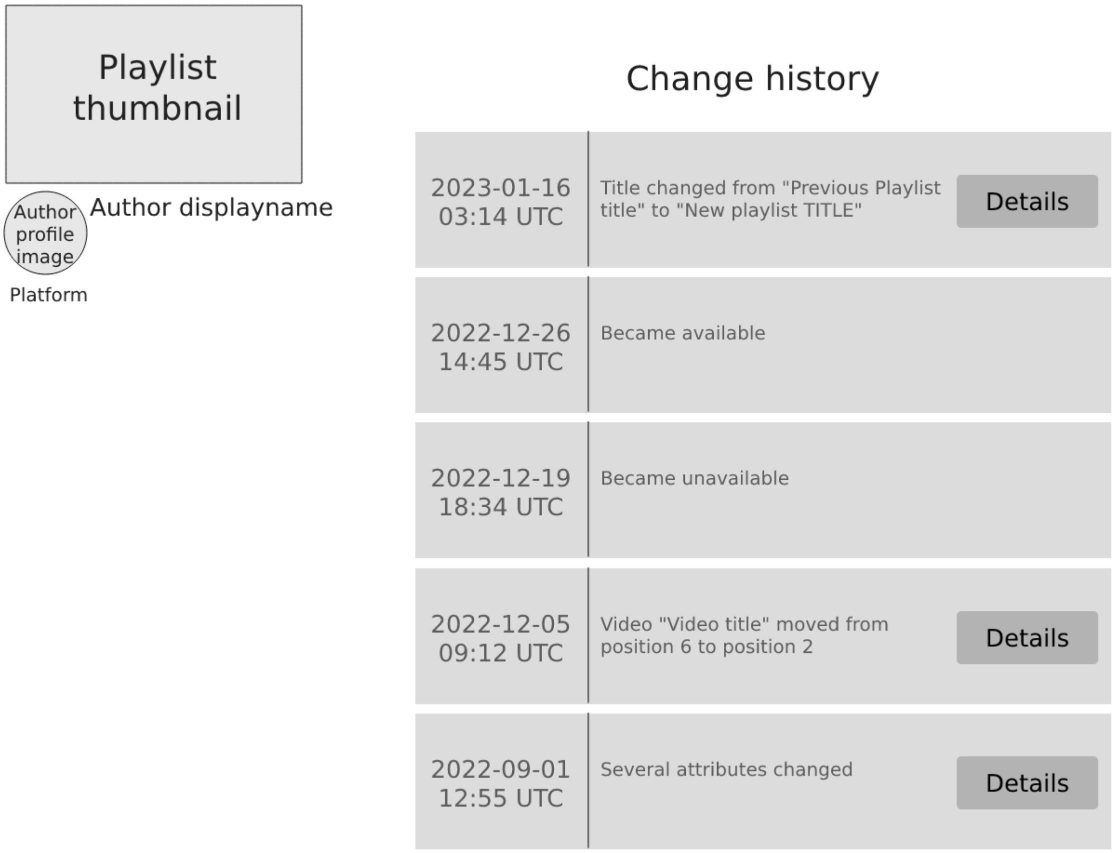


Figure 6. *An image of an editable playlist view in the web application.*

Upon clicking the "View history" button in the playlist view, the user will be taken to the playlist's change history page (demonstrated in Figure 7), where a timeline summarizing changes made to the playlist is displayed. If the changes can be concisely summarized, this summary will be displayed here, as is the case with the first four entries in Figure 7, starting from the top. However, the final visible entry (at the bottom) had more complex changes made and would require the user to click on the "Details" button to see what exactly was changed. Changes in the playlist's availability or privacy status are treated separately from other attribute changes and they can always be concisely summarized, which is why those entries have no "Details" button.

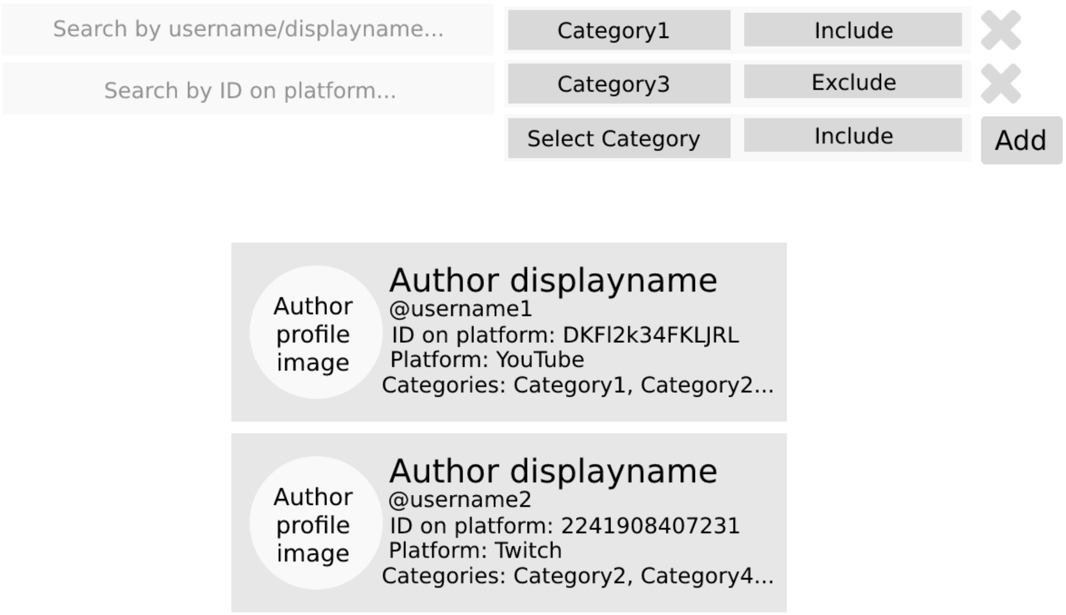


Change history

2023-01-16 03:14 UTC	Title changed from "Previous Playlist title" to "New playlist TITLE"
2022-12-26 14:45 UTC	Became available
2022-12-19 18:34 UTC	Became unavailable
2022-12-05 09:12 UTC	Video "Video title" moved from position 6 to position 2
2022-09-01 12:55 UTC	Several attributes changed

Figure 7. *An image of an overview of past changes to a playlist in the web application.*

Finally, there must be a way to search for and filter entities in the archive. Figure 8 displays a mock-up of what such a search page might look like for authors. There are options to search for authors by some of their attributes and filter by categories that they do or do not have. This search page and other like it will likely be significantly expanded during the project's development, to allow the search to be as customizable as possible. However, this will also likely cause some complications. For example, if a "search by URL" feature were implemented, it would need significant custom logic, especially since authors can often have multiple URLs, constructed based on different attributes. Searching by a custom-serialized attribute will be especially problematic, but should still be attempted – for example, searching by a video's title, which is a serialized dictionary of localization keys and matching title values, is a feature that would be nice to have.



Author displayname
@username1
ID on platform: DKFI2k34FKLJRL
Platform: YouTube
Categories: Category1, Category2...

Author displayname
@username2
ID on platform: 2241908407231
Platform: Twitch
Categories: Category2, Category4...

Figure 8. *An image of a search page for finding authors in the web application.*

4. Retrospective

Compared to the initial plans, the amount of functionality that was actually completed by the deadline is quite limited. The project's scope had to be reduced several times and even so, completing the project within the requirements of such a limited scope proved just barely possible. Several features were completely left out and some others are present in a limited capacity.

4.1 Short description of functionality

The project's most important functionality was implemented successfully: users can add videos to the archive, search for them, watch them and access their related data, e.g. title and comments. Adding playlists is also supported, but there is no way for the users to interact with the playlists themselves, other than seeing that the videos from the playlist are present in the archive after adding, even though the application does perform operations with the playlists behind the scenes. Added videos and playlists are regularly monitored for changes and updated.
Authors are added when a video, playlist or comment by that author is archived and users can see information about that author next to the relevant video, comment or playlist. However, no further author-related functionality is present, other than the ability to search for videos by their author's username.

Related to this main functionality, a permissions system was also implemented and integrated with an identity system. Every video in the archive has a privacy status. If the video is private, only administrators and users who have been granted access to that video can watch it and search for it. If the video is unlisted, only administrators can search for it, but anyone with a link to it can watch it. Finally, if the video is public, then anyone with a link to it can watch it and any logged in archive user can search for it.

If a user registers an account, an administrator must approve that account before it can be used. The administrator may also choose to grant some roles to the user. If the user has no roles, they can not submit anything to the archive, and can only search from the public videos in the archive. Any user with the helper role can submit a link to the archive, but that submission must then be approved by an administrator, before the link can be archived. When approving a submission, the administrator can also choose whether or not the user who submitted the link should be granted access to the archived entity. If a link is submitted by an administrator, the submission is approved automatically. An administrator can also choose to change a video's privacy status.

While all clients allow users to filter videos by category when searching them, only the ASP.NET Core MVC client supports managing categories and assigning them to videos. Also, the originally planned sub-category system was deemed needlessly complicated and thus the implemented system doesn't support categories being sub-categories of other categories.

In addition to these implemented or partially implemented features, there are also some initially planned features that had to be excluded completely due to time constraints. These include:

- Viewing and managing playlists.
- Submitting authors to the archive to have their uploads continually archived.
- Adding categories to authors and playlists.
- Adding ratings to videos, authors and playlists.
- Viewing the history of changes to an entity.
- Subscribing to authors and playlists to get notified of new videos.
- Notifying a user when an entity they added has its status changed, for example when it's deleted or changed to private on its platform of origin.
- Downloading subtitles, enabling users to select them on videos, and monitoring the subtitles for changes.
- Using non-polling methods such as webhooks or Pub/Sub to receive updates about changes to an entity.
- Archiving information about the game being played or discussed in a video, if the video has a game assigned to it.

Some of the differences between the initial plan and the final result can be seen by comparing the initial ERD (Appendix 2) to the final ERD (Appendix 3). The final ERD also has some entities color-coded to indicate that they are either not currently used or that they are not accessible via the user interface.

4.2 App architecture and noteworthy problems

4.2.1 Separation of platforms

Even though the application currently only deals with YouTube, it was important to design the application in such a way that support for other platforms could be added in the future without requiring large redesigns and without mixing up the logic for different platforms. This was accomplished by implementing a generic service layer project that performs functions common to all platforms and is also the only platform-related service used by the Web layer. Support for different platforms can be added as separate projects that register their functionality in the dependency injection engine to be used by the generic services.

For example, when a URL is submitted, the generic service for handling URL submissions gets a collection of URL submission handlers from the dependency injection engine. It then asks these handlers is any of them can be used to handle the submitted URL, and then lets the URL get handled by the first one that responds affirmatively.

In the previous example, if no handler capable of handling the URL is registered, an error is thrown. However, some platform-specific handlers can also be supplemental. When a video needs to be displayed in the user interface, the generic service fetches the video from the database and then checks if any handlers have been registered for handling the video presentation. In the case of a YouTube video, the registered handler is used to construct URLs out of the video and author IDs and selecting which thumbnail to use.

Thanks to this design, nothing in the Web layer or in the generic service project needs to perform any platform-specific logic, which can be cleanly separated into dedicated projects.

4.2.2 Background services

Many aspects of the application's functionality were implemented as services that run in the background, not as a result of user input in the form of a web request. Some of these background services regularly check for changes to videos and playlists to keep their archived counterparts up to date. Others are used to perform tasks that are triggered as the result of another action, but for one reason or another can't be performed within the context of that triggering action.

For example, when a video is submitted to the archive, time-consuming tasks such as

archiving that video's comments and downloading the video file shouldn't be performed within the context of executing the web request. Instead, the service handling the submission registers an event to fire after the basic metadata of the video have been saved to the database.
The relevant background services are subscribed to that event and start their tasks once the event is triggered. It's important that they start their work after the event has been fired, because before that the video isn't yet saved to the database and there's nothing to link the comment entities or the downloaded file to.

4.2.3 Data transfer objects and entity tracking

The application is separated into multiple layers that have different responsibilities and use different data transfer objects (DTOs) for performing their actions. As part of this architecture, the service layer doesn't directly use the database domain entities; it instead uses entities from the data access layer (DAL), which abstracts away database functionality. Two big benefits of this are the ability to only load required properties from the database and avoiding uncertainty about whether navigation properties and joined collections are loaded by either including or excluding them from the DAL-layer DTO contract.

However, it also comes with a downside – using DTOs in this manner causes Entity Framework Core, the database object-relational mapping framework used by this application, to not keep track of the fetched entities. This causes an issue if the fetched entities need to be updated or linked to other entities. Many hours and days during the development of the application at hand were spent investigating the Entity Framework Core change tracker to track down obscure bugs.

The solution for these issues ended up requiring writing a custom method for adding the fetched entities to the Entity Framework Core change tracker and later, when updating them or referencing them from other entities, checking if a matching entity with the same primary key was already being tracked and re-linking them, if so. Fortunately it was possible to keep this responsibility in the data access layer, but due to not having the foresight to make a concrete design decision about how linking entities should be approached on the service layer, some remnants of these issues remained in the application for a while and showed up later as unexpected bugs.

4.2.4 Issues with localization

One of the requirements for this project was that it had to have internationalization support. Partially because of this and partially because archiving translations is important in general, several entities in this application had to have properties that support translation. This includes video titles and descriptions, playlist titles and category titles.

Two approaches were considered for storing these properties: storing them in a separate database table or serializing them and storing them in a database column like other properties. Using a separate database table would've necessitated an extra join on many queries, which the author believed would introduce unnecessary complexity and potential performance issues. Additionally, according to initial plans, in most contexts it would've been necessary to load all translations at once. Thus, the serialization approach was chosen.

Each localizable property was stored as a dictionary where keys are cultures and the values are the translations in these cultures. However, the possibility of querying the database by these values was not considered when choosing the serialization approach. That was not required for most localizable properties, but searching for videos by video titles was an absolutely necessary feature.

This meant that the application required a way to perform a case-insensitive query over all video titles to see if any of their values contained a search string. Fortunately the application was using PostgreSQL as the database provider, which has a custom "JSONB" column type for storing such serializable values and also performing queries on them. Constructing such a query proved to be a ridiculously difficult task and a disproportionately large amount of time was spent on it.

At the moment of writing this, the queries appear to work properly. Yet even with the queries working, there remains a concern that this approach may cause performance issues when there are more videos in the database. Also, using PostgreSQL-specific features like this means that switching to a different database provider would likely require big changes to the application.

4.2.5 Video file access authorization

In addition to securing access to a video's data, access to the video file itself needed to be secured as well. After all, it wouldn't make sense to restrict someone from accessing a video's web page, but not place any security on the video file itself. This meant that the video file couldn't be served as a static file. It required a dedicated controller that could perform custom authorization logic based on the received request. The way the video file was served had to also support seeking to different points of time in the video and streaming the video file instead of sending all the data at once.

Once the controller was set up, serving the video file via the ASP.NET MVC web application worked flawlessly. However, the client applications written in JavaScript caused many problems on this front. The main issue was the fact that the HTML <video> element doesn't allow JavaScript to modify the requests it makes for fetching more of the video. This meant that the client applications wouldn't be able to set the authorization header for these requests, which is how the other requests of the JavaScript client applications were authenticated.

Even though the JavaScript application can't cause the video requests' authentication header to be changed, it can affect the cookies that are sent. The solution implemented in the end uses JavaScript to make an initial request to an API endpoint that causes the browser to set an HTTP-only cookie containing a JSON Web Token with a short lifetime. When the video file endpoint receives a request with that cookie set, it checks if the cookie is still valid and if it is, uses the information in the cookie to authenticate the user. Then, with the response that contains the video file data, it also replaces the cookie in the client browser with a new one. The JavaScript client application also periodically sends requests to the API endpoint that sets the cookie value, in case the video page is left open for a while without playing the video.

In order for that approach to work, the API application required re-configuration, because its default Cross-Origin Resource Sharing (CORS) policy allowed all request origins and that wasn't compatible with sending credentials via a cookie. Hence a separate CORS policy had to be configured for the video file endpoint and the JavaScript client applications' hostnames had to be explicitly registered as allowed for that policy.

4.2.6 Other issues

In addition to the aforementioned problems, there were a large amount of persistent issues that repeatedly obstructed development.

One of the sources of such problems was the token refreshing code in the JavaScript client application. Somehow the author managed to repeatedly make minor logic errors in these methods that only revealed themselves if the refresh occurred at a specific point in time.

Another source was configuring the ASP.NET Core web application. Whether it be stopping the application from returning HTML code from an API controller when authentication failed or spending hours debugging a controller's authorization logic, only to realize that the default cookie authentication scheme name for the default identity system isn't actually what's returned by "CookieAuthenticationDefaults.AuthenticationScheme", these issues took way longer to fix than they had any right to.

References

[1] *What is a webhook?* [Accessed: 26-02-2023]. URL: https://www.redhat.com/en/topics/automation/what-is-a-webhook.

[2] *What is a Pub/Sub?* [Accessed: 26-02-2023]. URL: https://cloud.google.com/pubsub/docs/overview.

[3] *YouTube Data API Reference - Videos*. [Accessed: 26-02-2023]. URL: https://developers.google.com/youtube/v3/docs/videos.

[4] *YouTube Data API Guides - Subscribe to Push Notifications*. [Accessed: 26-02-2023]. URL: https://developers.google.com/youtube/v3/guides/push_notifications.

[5] *Twitch API Docs - EventSub Subscription Types*. [Accessed: 26-02-2023]. URL: https://dev.twitch.tv/docs/eventsub/eventsub-subscription-types/.

Appendix 1 – Non-Exclusive License for Reproduction and Publication of a Graduation Thesis[1]

I Mikkel Paat

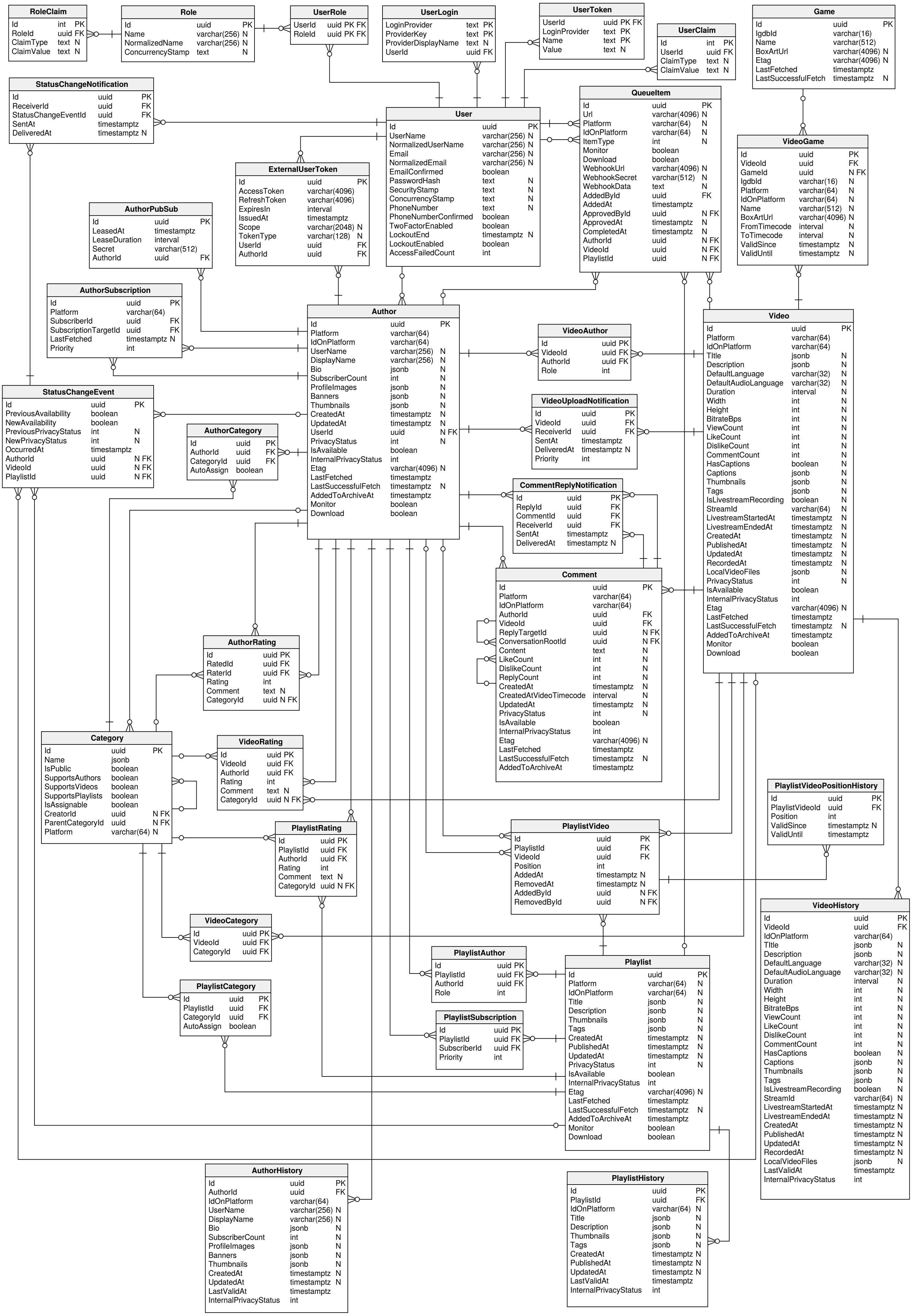
1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Online Video Archival System”, supervised by Andres Käver
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

06.06.2023

[1] The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – ERD (Initial plan)

This appendix can be found on the next page, due to its large size.



RoleClaim		
Id	int	PK
RoleId	uuid	FK
ClaimType	text	N
ClaimValue	text	N

Role		
Id	uuid	PK
Name	varchar(256)	N
NormalizedName	varchar(256)	N
ConcurrencyStamp	text	N

UserRole		
UserId	uuid	PK FK
RoleId	uuid	PK FK

UserLogin		
LoginProvider	text	PK
ProviderKey	text	PK
ProviderDisplayName	text	N
UserId	uuid	FK

UserToken		
UserId	uuid	PK FK
LoginProvider	text	PK
Name	text	PK
Value	text	N

UserClaim		
Id	int	PK
UserId	uuid	FK
ClaimType	text	N
ClaimValue	text	N

Game		
Id	uuid	PK
IgdbId	varchar(16)	
Name	varchar(512)	
BoxArtUrl	varchar(4096)	N
Etag	varchar(4096)	N
LastFetched	timestamptz	
LastSuccessfulFetch	timestamptz	N

StatusChangeNotification		
Id	uuid	PK
ReceiverId	uuid	FK
StatusChangeEventId	uuid	FK
SentAt	timestamptz	
DeliveredAt	timestamptz	N

User		
Id	uuid	PK
UserName	varchar(256)	N
NormalizedUserName	varchar(256)	N
Email	varchar(256)	N
NormalizedEmail	varchar(256)	N
EmailConfirmed	boolean	
PasswordHash	text	N
SecurityStamp	text	N
ConcurrencyStamp	text	N
PhoneNumber	text	N
PhoneNumberConfirmed	boolean	
TwoFactorEnabled	boolean	
LockoutEnd	timestamptz	N
LockoutEnabled	boolean	
AccessFailedCount	int	

QueueItem		
Id	uuid	PK
Url	varchar(4096)	N
Platform	varchar(64)	N
IdOnPlatform	varchar(64)	N
ItemType	int	N
Monitor	boolean	
Download	boolean	
WebhookUrl	varchar(4096)	N
WebhookSecret	varchar(512)	N
WebhookData	text	N
AddedById	uuid	FK
AddedAt	timestamptz	
ApprovedById	uuid	N FK
ApprovedAt	timestamptz	N
CompletedAt	timestamptz	N
AuthorId	uuid	N FK
VideoId	uuid	N FK
PlaylistId	uuid	N FK

ExternalUserToken		
Id	uuid	PK
AccessToken	varchar(4096)	
RefreshToken	varchar(4096)	
ExpiresIn	interval	
IssuedAt	timestamptz	
Scope	varchar(2048)	N
TokenType	varchar(128)	N
UserId	uuid	FK
AuthorId	uuid	FK

VideoGame		
Id	uuid	PK
VideoId	uuid	FK
GameId	uuid	N FK
IgdbId	varchar(16)	N
Platform	varchar(64)	N
IdOnPlatform	varchar(64)	N
Name	varchar(512)	N
BoxArtUrl	varchar(4096)	N
FromTimecode	interval	N
ToTimecode	interval	N
ValidSince	timestamptz	N
ValidUntil	timestamptz	N

AuthorPubSub		
Id	uuid	PK
LeasedAt	timestamptz	
LeaseDuration	interval	
Secret	varchar(512)	
AuthorId	uuid	FK

AuthorSubscription		
Id	uuid	PK
Platform	varchar(64)	
SubscriberId	uuid	FK
SubscriptionTargetId	uuid	FK
LastFetched	timestamptz	N
Priority	int	

Author		
Id	uuid	PK
Platform	varchar(64)	
IdOnPlatform	varchar(64)	
UserName	varchar(256)	N
DisplayName	varchar(256)	N
Bio	jsonb	N
SubscriberCount	int	N
ProfileImages	jsonb	N
Banners	jsonb	N
Thumbnails	jsonb	N
CreatedAt	timestamptz	N
UpdatedAt	timestamptz	N
UserId	uuid	N FK
PrivacyStatus	int	N
IsAvailable	boolean	
InternalPrivacyStatus	int	
Etag	varchar(4096)	N
LastFetched	timestamptz	
LastSuccessfulFetch	timestamptz	N
AddedToArchiveAt	timestamptz	
Monitor	boolean	
Download	boolean	

VideoAuthor		
Id	uuid	PK
VideoId	uuid	FK
AuthorId	uuid	FK
Role	int	

Video		
Id	uuid	PK
Platform	varchar(64)	
IdOnPlatform	varchar(64)	
TItle	jsonb	N
Description	jsonb	N
DefaultLanguage	varchar(32)	N
DefaultAudioLanguage	varchar(32)	N
Duration	interval	N
Width	int	N
Height	int	N
BitrateBps	int	N
ViewCount	int	N
LikeCount	int	N
DislikeCount	int	N
CommentCount	int	N
HasCaptions	boolean	N
Captions	jsonb	N
Thumbnails	jsonb	N
Tags	jsonb	N
IsLivestreamRecording	boolean	N
StreamId	varchar(64)	N
LivestreamStartedAt	timestamptz	N
LivestreamEndedAt	timestamptz	N
CreatedAt	timestamptz	N
PublishedAt	timestamptz	N
UpdatedAt	timestamptz	N
RecordedAt	timestamptz	N
LocalVideoFiles	jsonb	N
PrivacyStatus	int	N
IsAvailable	boolean	
InternalPrivacyStatus	int	
Etag	varchar(4096)	N
LastFetched	timestamptz	
LastSuccessfulFetch	timestamptz	N
AddedToArchiveAt	timestamptz	
Monitor	boolean	
Download	boolean	

StatusChangeEvent		
Id	uuid	PK
PreviousAvailability	boolean	
NewAvailability	boolean	
PreviousPrivacyStatus	int	N
NewPrivacyStatus	int	N
OccurredAt	timestamptz	
AuthorId	uuid	N FK
VideoId	uuid	N FK
PlaylistId	uuid	N FK

VideoUploadNotification		
Id	uuid	PK
VideoId	uuid	FK
ReceiverId	uuid	FK
SentAt	timestamptz	
DeliveredAt	timestamptz	N
Priority	int	

AuthorCategory		
Id	uuid	PK
AuthorId	uuid	FK
CategoryId	uuid	FK
AutoAssign	boolean	

CommentReplyNotification		
Id	uuid	PK
ReplyId	uuid	FK
CommentId	uuid	FK
ReceiverId	uuid	FK
SentAt	timestamptz	
DeliveredAt	timestamptz	N

Comment		
Id	uuid	PK
Platform	varchar(64)	
IdOnPlatform	varchar(64)	
AuthorId	uuid	FK
VideoId	uuid	FK
ReplyTargetId	uuid	N FK
ConversationRootId	uuid	N FK
Content	text	N
LikeCount	int	N
DislikeCount	int	N
ReplyCount	int	N
CreatedAt	timestamptz	N
CreatedAtVideoTimecode	interval	N
UpdatedAt	timestamptz	N
PrivacyStatus	int	N
IsAvailable	boolean	
InternalPrivacyStatus	int	
Etag	varchar(4096)	N
LastFetched	timestamptz	
LastSuccessfulFetch	timestamptz	N
AddedToArchiveAt	timestamptz	

AuthorRating		
Id	uuid	PK
RatedId	uuid	FK
RaterId	uuid	FK
Rating	int	
Comment	text	N
CategoryId	uuid	N FK

Category		
Id	uuid	PK
Name	jsonb	
IsPublic	boolean	
SupportsAuthors	boolean	
SupportsVideos	boolean	
SupportsPlaylists	boolean	
IsAssignable	boolean	
CreatorId	uuid	N FK
ParentCategoryId	uuid	N FK
Platform	varchar(64)	N

VideoRating		
Id	uuid	PK
VideoId	uuid	FK
AuthorId	uuid	FK
Rating	int	
Comment	text	N
CategoryId	uuid	N FK

PlaylistVideoPositionHistory		
Id	uuid	PK
PlaylistVideoId	uuid	FK
Position	int	
ValidSince	timestamptz	N
ValidUntil	timestamptz	

PlaylistRating		
Id	uuid	PK
PlaylistId	uuid	FK
AuthorId	uuid	FK
Rating	int	
Comment	text	N
CategoryId	uuid	N FK

PlaylistVideo		
Id	uuid	PK
PlaylistId	uuid	FK
VideoId	uuid	FK
Position	int	
AddedAt	timestamptz	N
RemovedAt	timestamptz	N
AddedById	uuid	N FK
RemovedById	uuid	N FK

VideoHistory		
Id	uuid	PK
VideoId	uuid	FK
IdOnPlatform	varchar(64)	
TItle	jsonb	N
Description	jsonb	N
DefaultLanguage	varchar(32)	N
DefaultAudioLanguage	varchar(32)	N
Duration	interval	N
Width	int	N
Height	int	N
BitrateBps	int	N
ViewCount	int	N
LikeCount	int	N
DislikeCount	int	N
CommentCount	int	N
HasCaptions	boolean	N
Captions	jsonb	N
Thumbnails	jsonb	N
Tags	jsonb	N
IsLivestreamRecording	boolean	N
StreamId	varchar(64)	N
LivestreamStartedAt	timestamptz	N
LivestreamEndedAt	timestamptz	N
CreatedAt	timestamptz	N
PublishedAt	timestamptz	N
UpdatedAt	timestamptz	N
RecordedAt	timestamptz	N
LocalVideoFiles	jsonb	N
LastValidAt	timestamptz	
InternalPrivacyStatus	int	

VideoCategory		
Id	uuid	PK
VideoId	uuid	FK
CategoryId	uuid	FK

PlaylistAuthor		
Id	uuid	PK
PlaylistId	uuid	FK
AuthorId	uuid	FK
Role	int	

Playlist		
Id	uuid	PK
Platform	varchar(64)	N
IdOnPlatform	varchar(64)	N
Title	jsonb	N
Description	jsonb	N
Thumbnails	jsonb	N
Tags	jsonb	N
CreatedAt	timestamptz	N
PublishedAt	timestamptz	N
UpdatedAt	timestamptz	N
PrivacyStatus	int	N
IsAvailable	boolean	
InternalPrivacyStatus	int	
Etag	varchar(4096)	N
LastFetched	timestamptz	
LastSuccessfulFetch	timestamptz	N
AddedToArchiveAt	timestamptz	
Monitor	boolean	
Download	boolean	

PlaylistCategory		
Id	uuid	PK
PlaylistId	uuid	FK
CategoryId	uuid	FK
AutoAssign	boolean	

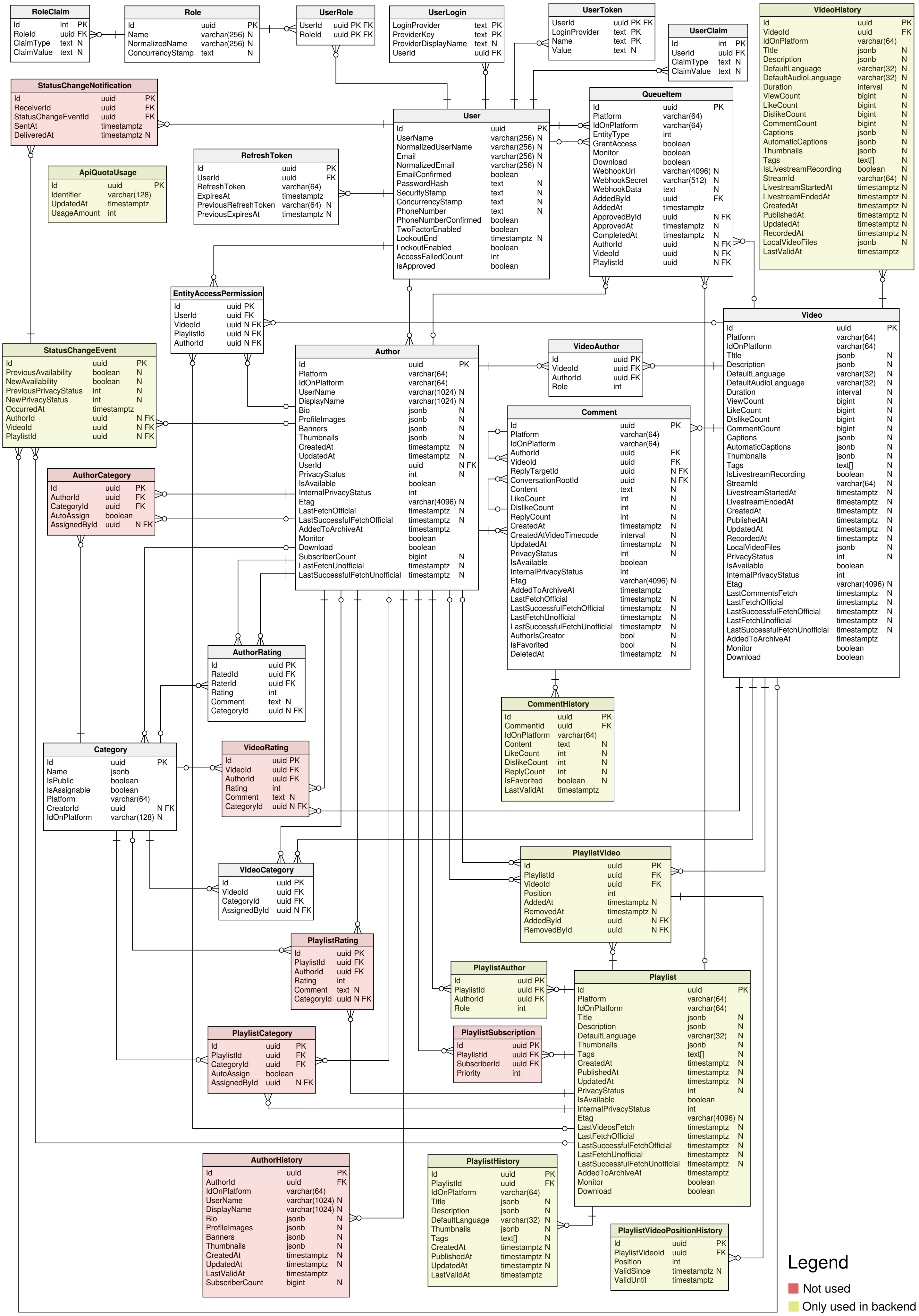
PlaylistSubscription		
Id	uuid	PK
PlaylistId	uuid	FK
SubscriberId	uuid	FK
Priority	int	

AuthorHistory		
Id	uuid	PK
AuthorId	uuid	FK
IdOnPlatform	varchar(64)	
UserName	varchar(256)	N
DisplayName	varchar(256)	N
Bio	jsonb	N
SubscriberCount	int	N
ProfileImages	jsonb	N
Banners	jsonb	N
Thumbnails	jsonb	N
CreatedAt	timestamptz	N
UpdatedAt	timestamptz	N
LastValidAt	timestamptz	
InternalPrivacyStatus	int	

PlaylistHistory		
Id	uuid	PK
PlaylistId	uuid	FK
IdOnPlatform	varchar(64)	N
Title	jsonb	N
Description	jsonb	N
Thumbnails	jsonb	N
Tags	jsonb	N
CreatedAt	timestamptz	N
PublishedAt	timestamptz	N
UpdatedAt	timestamptz	N
LastValidAt	timestamptz	
InternalPrivacyStatus	int	

Appendix 3 – ERD (Final)

This appendix can be found on the next page, due to its large size.



RoleClaim		
Id	int	PK
RoleId	uuid	FK
ClaimType	text	N
ClaimValue	text	N

Role		
Id	uuid	PK
Name	varchar(256)	N
NormalizedName	varchar(256)	N
ConcurrencyStamp	text	N

UserRole		
UserId	uuid	PK FK
RoleId	uuid	PK FK

UserLogin		
LoginProvider	text	PK
ProviderKey	text	PK
ProviderDisplayName	text	N
UserId	uuid	FK

UserToken		
UserId	uuid	PK FK
LoginProvider	text	PK
Name	text	PK
Value	text	N

UserClaim		
Id	int	PK
UserId	uuid	FK
ClaimType	text	N
ClaimValue	text	N

VideoHistory		
Id	uuid	PK
VideoId	uuid	FK
IdOnPlatform	varchar(64)	
TItle	jsonb	N
Description	jsonb	N
DefaultLanguage	varchar(32)	N
DefaultAudioLanguage	varchar(32)	N
Duration	interval	N
ViewCount	bigint	N
LikeCount	bigint	N
DislikeCount	bigint	N
CommentCount	bigint	N
Captions	jsonb	N
AutomaticCaptions	jsonb	N
Thumbnails	jsonb	N
Tags	text[]	N
IsLivestreamRecording	boolean	N
StreamId	varchar(64)	N
LivestreamStartedAt	timestamptz	N
LivestreamEndedAt	timestamptz	N
CreatedAt	timestamptz	N
PublishedAt	timestamptz	N
UpdatedAt	timestamptz	N
RecordedAt	timestamptz	N
LocalVideoFiles	jsonb	N
LastValidAt	timestamptz	

StatusChangeNotification		
Id	uuid	PK
ReceiverId	uuid	FK
StatusChangeEventId	uuid	FK
SentAt	timestamptz	
DeliveredAt	timestamptz	N

User		
Id	uuid	PK
UserName	varchar(256)	N
NormalizedUserName	varchar(256)	N
Email	varchar(256)	N
NormalizedEmail	varchar(256)	N
EmailConfirmed	boolean	
PasswordHash	text	N
SecurityStamp	text	N
ConcurrencyStamp	text	N
PhoneNumber	text	N
PhoneNumberConfirmed	boolean	
TwoFactorEnabled	boolean	
LockoutEnd	timestamptz	N
LockoutEnabled	boolean	
AccessFailedCount	int	
IsApproved	boolean	

QueueItem		
Id	uuid	PK
Platform	varchar(64)	
IdOnPlatform	varchar(64)	
EntityType	int	
GrantAccess	boolean	
Monitor	boolean	
Download	boolean	
WebhookUrl	varchar(4096)	N
WebhookSecret	varchar(512)	N
WebhookData	text	N
AddedById	uuid	FK
AddedAt	timestamptz	
ApprovedById	uuid	N FK
ApprovedAt	timestamptz	N
CompletedAt	timestamptz	N
AuthorId	uuid	N FK
VideoId	uuid	N FK
PlaylistId	uuid	N FK

ApiQuotaUsage		
Id	uuid	PK
Identifier	varchar(128)	
UpdatedAt	timestamptz	
UsageAmount	int	

RefreshToken		
Id	uuid	PK
UserId	uuid	FK
RefreshToken	varchar(64)	
ExpiresAt	timestamptz	
PreviousRefreshToken	varchar(64)	N
PreviousExpiresAt	timestamptz	N

EntityAccessPermission		
Id	uuid	PK
UserId	uuid	FK
VideoId	uuid	N FK
PlaylistId	uuid	N FK
AuthorId	uuid	N FK

StatusChangeEvent		
Id	uuid	PK
PreviousAvailability	boolean	N
NewAvailability	boolean	N
PreviousPrivacyStatus	int	N
NewPrivacyStatus	int	N
OccurredAt	timestamptz	
AuthorId	uuid	N FK
VideoId	uuid	N FK
PlaylistId	uuid	N FK

Author		
Id	uuid	PK
Platform	varchar(64)	
IdOnPlatform	varchar(64)	
UserName	varchar(1024)	N
DisplayName	varchar(1024)	N
Bio	jsonb	N
ProfileImages	jsonb	N
Banners	jsonb	N
Thumbnails	jsonb	N
CreatedAt	timestamptz	N
UpdatedAt	timestamptz	N
UserId	uuid	N FK
PrivacyStatus	int	N
IsAvailable	boolean	
InternalPrivacyStatus	int	
Etag	varchar(4096)	N
LastFetchOfficial	timestamptz	N
LastSuccessfulFetchOfficial	timestamptz	N
AddedToArchiveAt	timestamptz	
Monitor	boolean	
Download	boolean	
SubscriberCount	bigint	N
LastFetchUnofficial	timestamptz	N
LastSuccessfulFetchUnofficial	timestamptz	N

VideoAuthor		
Id	uuid	PK
VideoId	uuid	FK
AuthorId	uuid	FK
Role	int	

Video		
Id	uuid	PK
Platform	varchar(64)	
IdOnPlatform	varchar(64)	
TItle	jsonb	N
Description	jsonb	N
DefaultLanguage	varchar(32)	N
DefaultAudioLanguage	varchar(32)	N
Duration	interval	N
ViewCount	bigint	N
LikeCount	bigint	N
DislikeCount	bigint	N
CommentCount	bigint	N
Captions	jsonb	N
AutomaticCaptions	jsonb	N
Thumbnails	jsonb	N
Tags	text[]	N
IsLivestreamRecording	boolean	N
StreamId	varchar(64)	N
LivestreamStartedAt	timestamptz	N
LivestreamEndedAt	timestamptz	N
CreatedAt	timestamptz	N
PublishedAt	timestamptz	N
UpdatedAt	timestamptz	N
RecordedAt	timestamptz	N
LocalVideoFiles	jsonb	N
PrivacyStatus	int	N
IsAvailable	boolean	
InternalPrivacyStatus	int	
Etag	varchar(4096)	N
LastCommentsFetch	timestamptz	N
LastFetchOfficial	timestamptz	N
LastSuccessfulFetchOfficial	timestamptz	N
LastFetchUnofficial	timestamptz	N
LastSuccessfulFetchUnofficial	timestamptz	N
AddedToArchiveAt	timestamptz	
Monitor	boolean	
Download	boolean	

Comment		
Id	uuid	PK
Platform	varchar(64)	
IdOnPlatform	varchar(64)	
AuthorId	uuid	FK
VideoId	uuid	FK
ReplyTargetId	uuid	N FK
ConversationRootId	uuid	N FK
Content	text	N
LikeCount	int	N
DislikeCount	int	N
ReplyCount	int	N
CreatedAt	timestamptz	N
CreatedAtVideoTimecode	interval	N
UpdatedAt	timestamptz	N
PrivacyStatus	int	N
IsAvailable	boolean	
InternalPrivacyStatus	int	
Etag	varchar(4096)	N
AddedToArchiveAt	timestamptz	
LastFetchOfficial	timestamptz	N
LastSuccessfulFetchOfficial	timestamptz	N
LastFetchUnofficial	timestamptz	N
LastSuccessfulFetchUnofficial	timestamptz	N
AuthorIsCreator	bool	N
IsFavorited	bool	N
DeletedAt	timestamptz	N

AuthorCategory		
Id	uuid	PK
AuthorId	uuid	FK
CategoryId	uuid	FK
AutoAssign	boolean	
AssignedById	uuid	N FK

AuthorRating		
Id	uuid	PK
RatedId	uuid	FK
RaterId	uuid	FK
Rating	int	
Comment	text	N
CategoryId	uuid	N FK

CommentHistory		
Id	uuid	PK
CommentId	uuid	FK
IdOnPlatform	varchar(64)	
Content	text	N
LikeCount	int	N
DislikeCount	int	N
ReplyCount	int	N
IsFavorited	boolean	N
LastValidAt	timestamptz	

Category		
Id	uuid	PK
Name	jsonb	
IsPublic	boolean	
IsAssignable	boolean	
Platform	varchar(64)	
CreatorId	uuid	N FK
IdOnPlatform	varchar(128)	N

VideoRating		
Id	uuid	PK
VideoId	uuid	FK
AuthorId	uuid	FK
Rating	int	
Comment	text	N
CategoryId	uuid	N FK

PlaylistVideo		
Id	uuid	PK
PlaylistId	uuid	FK
VideoId	uuid	FK
Position	int	
AddedAt	timestamptz	N
RemovedAt	timestamptz	N
AddedById	uuid	N FK
RemovedById	uuid	N FK

VideoCategory		
Id	uuid	PK
VideoId	uuid	FK
CategoryId	uuid	FK
AssignedById	uuid	N FK

PlaylistRating		
Id	uuid	PK
PlaylistId	uuid	FK
AuthorId	uuid	FK
Rating	int	
Comment	text	N
CategoryId	uuid	N FK

PlaylistAuthor		
Id	uuid	PK
PlaylistId	uuid	FK
AuthorId	uuid	FK
Role	int	

Playlist		
Id	uuid	PK
Platform	varchar(64)	
IdOnPlatform	varchar(64)	
Title	jsonb	N
Description	jsonb	N
DefaultLanguage	varchar(32)	N
Thumbnails	jsonb	N
Tags	text[]	N
CreatedAt	timestamptz	N
PublishedAt	timestamptz	N
UpdatedAt	timestamptz	N
PrivacyStatus	int	N
IsAvailable	boolean	
InternalPrivacyStatus	int	
Etag	varchar(4096)	N
LastVideosFetch	timestamptz	N
LastFetchOfficial	timestamptz	N
LastSuccessfulFetchOfficial	timestamptz	N
LastFetchUnofficial	timestamptz	N
LastSuccessfulFetchUnofficial	timestamptz	N
AddedToArchiveAt	timestamptz	
Monitor	boolean	
Download	boolean	

PlaylistCategory		
Id	uuid	PK
PlaylistId	uuid	FK
CategoryId	uuid	FK
AutoAssign	boolean	
AssignedById	uuid	N FK

PlaylistSubscription		
Id	uuid	PK
PlaylistId	uuid	FK
SubscriberId	uuid	FK
Priority	int	

AuthorHistory		
Id	uuid	PK
AuthorId	uuid	FK
IdOnPlatform	varchar(64)	
UserName	varchar(1024)	N
DisplayName	varchar(1024)	N
Bio	jsonb	N
ProfileImages	jsonb	N
Banners	jsonb	N
Thumbnails	jsonb	N
CreatedAt	timestamptz	N
UpdatedAt	timestamptz	N
LastValidAt	timestamptz	
SubscriberCount	bigint	N

PlaylistHistory		
Id	uuid	PK
PlaylistId	uuid	FK
IdOnPlatform	varchar(64)	
Title	jsonb	N
Description	jsonb	N
DefaultLanguage	varchar(32)	N
Thumbnails	jsonb	N
Tags	text[]	N
CreatedAt	timestamptz	N
PublishedAt	timestamptz	N
UpdatedAt	timestamptz	N
LastValidAt	timestamptz	

PlaylistVideoPositionHistory		
Id	uuid	PK
PlaylistVideoId	uuid	FK
Position	int	
ValidSince	timestamptz	N
ValidUntil	timestamptz	

Legend

🟥 Not used

🟨 Only used in backend