



MediaVerse

A universe of media assets
and co-creation opportunities

D6.3

Core Framework, Decentralized Communication and Content Exchange v2

Project Title	MediaVerse
Contract No.	957252
Instrument	Innovation Action
Thematic Priority	ICT-44-2020 Next Generation Media
Start of Project	1 October 2020
Duration	36 months

Deliverable title	Core Framework, Decentralized Communication and Content Exchange v2
Deliverable number	D6.3
Deliverable version	V1.0
Previous version(s)	D6.2
Contractual Date of delivery	30.09.2022
Actual Date of delivery	11.10.2022
Nature of deliverable	Other
Dissemination level	Public
Partner Responsible	ATOS
Author(s)	Antonio Calvo García del Valle, Rubén Ramiro (ATOS), Panagiotis Galopoulos (CERTH), Tasos Lampropoulos, Spyros Papafragkos (ATC)
Reviewer(s)	Manos Schinas (CERTH), Andrea Cavallaro, Marco Giovanelli, Marco Saltarella (FINCONS)
EC Project Officer	Luis Eduardo Martinez Lafuente

Abstract	This deliverable presents a functional and technical description of the core elements of a MediaVerse node. The updated versions of the Digital Asset Management and Decentralized Framework are described in detail. Screenshots of the status of the user interface are also provided.
Keywords	Digital Asset Management, Decentralized framework, Dashboard, User Interface, Federation Shared Dataspace, Federated Search.

Copyright

© Copyright 2022 MediaVerse Consortium

This document may not be copied, reproduced, or modified in whole or in part for any purpose without written permission from the MediaVerse Consortium. In addition to such written permission to copy, reproduce, or modify this document in whole or part, an acknowledgement of the authors of the document and all applicable portions of the copyright notice must be clearly referenced.

All rights reserved.



MediaVerse is an H2020 Innovation Project co-financed by the EC under Grant Agreement ID: 957252. The content of this document is © the author(s). For further information, visit mediaverse-project.eu.

Revision History

VERSION	DATE	MODIFIED BY	COMMENTS
V0.1	22/07/2022	Antonio Calvo (ATOS)	First Draft Table of Content
V0.2	12/08/2022	Antonio Calvo (ATOS)	Added Decentralized Framework for MediaVerse Communication section
V0.3	26/08/2022	Marco Giovanelli (FINCONS)	Added Digital Rights Negotiation section
V0.4	02/09/2022	Rubén Ramiro (ATOS)	Added User Interface section
V0.5	05/09/2022	Tasos Lampropoulos (ATC)	Added Digit Asset Management and Component Development, Integration and Deployment sections
V0.6	06/09/2022	Rodrigo Peces, Rubén Ramiro (ATOS)	Added subsection Project to User Interface section and Final draft
V0.7	13/09/2022	Manos Schinas (CERTH), Marco Giovanelli (FINCONS)	Review and Feedback
V0.8	19/09/2022	Tasos Lampropoulos (ATC)	Updated 2.2.1 and 2.2.2 from 1 st Review comments.
V0.9	27/09/2022	Rubén Ramiro, Antonio Calvo (ATOS)	Updated doc based on feedback.
V1.0	29/09/2022	Evangelia Kartounidou; Akis Papadopoulos (CERTH)	QA and final document

Glossary

ABBREVIATION	MEANING
AD	Administrative Domains
API	Application Programming Interface
CL	Copyright License
DAG	Directed Acyclic Graph
DAM	Digital Archive Management
DARM	Digital Asset Rights Management
DOA	Description Of the Action
HTTP	Hyper Text Transfer Protocol
IP	Internet Protocol
IPFS	Inter Planetary File System
IPR	Intellectual Property Rights
MV	MediaVerse
NDD	Near Duplicate Detection
OD	Ownership Deed
SLC	Smart Legal Contract
UI	User Interface
WCAG	Web Content Accessibility Guidelines
WP	Work Package



MediaVerse is an H2020 Innovation Project co-financed by the EC under Grant Agreement ID: 957252. The content of this document is © the author(s). For further information, visit mediaverse-project.eu.

Table of Contents

Revision History	3
Glossary	3
Table of Contents	4
Index of Figures	6
Executive Summary	8
1 Introduction.....	9
2 Digital Asset Management	11
2.1 Functional Description.....	11
2.2 Technical Description	12
2.2.1 Conceptual Architecture.....	12
2.2.2 Technical Implementation.....	13
2.2.3 Secure Communication	15
3 Dashboard – UI	16
3.1 Functional Description.....	16
3.1.1 Login	16
3.1.2 Dashboard	17
3.1.3 Profile.....	19
3.1.4 Language.....	19
3.1.5 Help.....	20
3.1.6 Upload	21
3.1.7 MyAssets.....	22
3.1.8 Local and Federated Search.....	28
3.1.9 Projects.....	30
3.2 Technical Description	32
4 Decentralized Framework for MediaVerse Communication.....	34
4.1 Introduction.....	34
4.2 Technical Description	35
4.2.1 Basic Infrastructure and Frameworks.....	35
4.2.2 External Communication	36
4.2.3 NDD and Future Services Integration	38
4.2.4 SLC Shared Dataspace.....	39
4.2.5 Internal Architecture	41

- 4.3 Functional Description..... 42
 - 4.4 Deployment 43
- 5 Component Development, Integration and Deployment 45
 - 5.1 Feature Development Process 45
 - 5.2 Integration 47
 - 5.3 Deployment 48
- 6 Conclusion and Next Steps 49

Index of Figures

Figure 1: MediaVerse Conceptual architecture.....	9
Figure 2: Main DAM functionalities.....	11
Figure 3: Swagger of DAM API.....	12
Figure 4: The DAM Service	13
Figure 5: Code structure	14
Figure 6: Create Account Window.....	16
Figure 7: MV Default Tab (Dashboard).....	17
Figure 8: Social Media Login Section	17
Figure 9: Twitter redirection to Login.....	18
Figure 10: Invite to MV card	18
Figure 11: Notification Card	18
Figure 12: Profile (Top Bar).....	19
Figure 13: User info	19
Figure 14: Billing Information / Wallet.....	19
Figure 15: Languages Available	20
Figure 16: Left Bar	20
Figure 17: Dashboard Section Description	20
Figure 18: Help Functionality.....	21
Figure 19: Upload Section	21
Figure 20: Successfully Upload	22
Figure 21: Failed Upload.....	22
Figure 22: My Assets view	23
Figure 23: Filter Controls	23
Figure 24: Three.js for 3D models	24
Figure 25: Captions for video	24
Figure 26: Disturbing Content Moderation Example	25
Figure 27: Share via Twitter.....	25
Figure 28: View of an asset.....	26
Figure 29: Languages	27
Figure 30: Labels to add manually moderation to the asset.....	27
Figure 31: License Tab	27
Figure 32: License Advisor	27
Figure 33: Successful CC license registration.....	28
Figure 34: Failed registration.....	28
Figure 35: Projects Tab	28
Figure 36: Local Search.....	29
Figure 37: External Search, CERTH node	29
Figure 38: External Search, ATC node	30
Figure 39: Project description.	30
Figure 40: Create New Project.....	30
Figure 41: Project created	30
Figure 42: Project Details Description	31
Figure 43: Project assets.....	32

Figure 44: Adding and removing users from the project.	32
Figure 45: Decentralized Framework Docker infrastructure.....	35
Figure 46: Inter-node pub-sub implementation.....	36
Figure 47: Grouped Federated search result example.....	37
Figure 48: NDD service and Federated Search integration	38
Figure 49 Smart Legal Contracts examples	39
Figure 50 SLC storage in and retrieval from IPFS.....	40
Figure 51: IPFS_API Internal architecture.....	41
Figure 52: IPFS_API documentation - swagger interface	42
Figure 53: IPFS_API interactive documentation detail.....	43
Figure 54: Federated search results in the UI	43
Figure 55: IPFS AP successful initialization logs.....	44
Figure 56. Docker Containers	45
Figure 57: The Features board.....	45
Figure 58: Board of Development issues.....	46
Figure 59: Board of Development issues.....	46
Figure 60: A view of docker hub repository	47

Executive Summary

This is the second release of the Core Framework, Decentralized Communication and Content Exchange following the initial version of D6.1¹ that was delivered in January 2022. In the first release, we reported the objectives of this WP and a number of architecture diagrams. The objective of this new version is to report on the achieved progress with respect to this initial architecture, as well as on the challenges and devised solutions.

The deliverable is divided into three main sections:

1. **The core framework of a node, called the DAM in the context of the project:** This element provides the basis for the rest of the platform services, facilitating the storage and publication of data, as well as the access and communication between the different developed micro-services.
2. **The User Interface:** This report describes the latest status of the user interface that enables users to access the MediaVerse nodes in an intuitive way. The technical implementation of the interface is also described in order to show the challenges faced in the development of the components.
3. **The decentralized framework:** This framework allows interaction between MediaVerse nodes without a master node controlling them. The auto-discovery mechanism that allows any node in the decentralized MediaVerse network to learn about new and existing nodes is also explained. The report also describes how the federated content search service has been integrated with the interface and the copy detection service, and delineates the possibility of integrating new components such as the copyright negotiation service and the recommendation service.

¹https://mediaverse-project.eu/wp-content/uploads/2022/02/MediaVerse_D6.1_Core-Framework_Decentralized-Communication_Content-Exchange_v1.0.pdf

1 Introduction

This deliverable is the second release of the Core-Framework Decentralized Communication Content-Exchange following the D6.1 - v1.0 release², delivered in January 2022.

The WP6 objectives remain the same: “WP6 develops and integrates into MediaVerse nodes the functionality for creation, storage, editing, exchange, and publication of content. Also, it is responsible for implementing the architecture for deploying and managing MediaVerse nodes and for the communication layer that makes possible the interaction between nodes. Additionally, it implements a User Interface that makes it possible for the user to access the different services available in MediaVerse in a user-friendly and intuitive way.”

Figure 1, which has been extracted from D2.2 - Conceptual Design of the MediaVerse Framework³, depicts the global architecture of the MediaVerse federated network, illustrating for the sake of simplicity a pair of two nodes, the components integrated as external services and tools and the interactions between them.

The MediaVerse ecosystem is a federation of MediaVerse Nodes. More formally, a MediaVerse ecosystem is made up of a set of Administrative Domains (ADs), where an AD has full responsibility of the administration of owned resources (e.g., digital assets, users, policies, HW/SW components). In the MediaVerse federation, each AD is autonomously managing its resources even if the AD in a MediaVerse federation cooperate according to a set of common rules governing the cooperation. This MediaVerse node federation offers additional services such as asset search between different media providers or automatic license negotiation between media providers.

A MediaVerse node is intended to be deployed universally, i.e., on hardware with very different capabilities. Therefore, there is a differentiation between micro-services deployed inside the node and external tools. These external micro-services/tools have special hardware requirements or are tools that for security reasons should not be freely used by any node without the authorization of their owners. From the micro services presented inside every node, this document will explain in detail the DAM, the Federated search, and the User Interface.

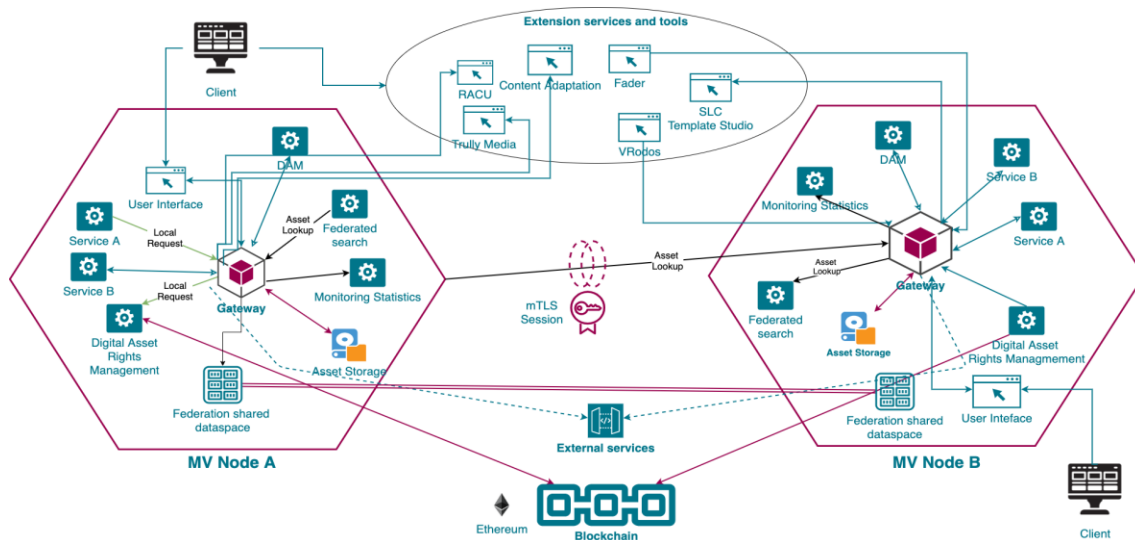


Figure 1: MediaVerse Conceptual architecture

²https://mediaverse-project.eu/wp-content/uploads/2022/02/MediaVerse_D6.1_Core-Framework_Decentralized-Communication_Content-Exchange_v1.0.pdf

³https://mediaverse-project.eu/wp-content/uploads/2021/10/MediaVerse_D2.2_Conceptual-Design-of-the-MediaVerse-Framework.pdf

One of the basic pillars of the MediaVerse is decentralization. Technically, the Internet is still at the dawn of decentralized web 3.0 and only some frameworks that can take decentralization into account are beginning to emerge. This document details the actions that have been taken to facilitate making MediaVerse a decentralized platform. During the early phases of this decentralized development, both technical and legal challenges were encountered. Hence, this document will attempt to highlight these challenges, the solutions that have been devised, and the open issues.

In the context of a single platform based on micro-services and developed by several partners, constant integration and testing is necessary. For that reason, the mechanisms carried out during the project for a continuous integration will be described.

2 Digital Asset Management

The following section is focusing on the Data Asset Management (DAM) component of the MediaVerse node. The DAM is a central element of the MediaVerse node and participates in almost all interactions between the components of the ecosystem. It is responsible for implementing the business logic of specific workflows, namely user management, access rights and local search and retrieval, which are described in subsequent sections.

2.1 Functional Description

The DAM is responsible for the management of digital multimedia objects. Moreover, it acts as the access point for all incoming calls to the MV Node by external clients (e.g., web application running in browser, distributed nodes, etc.). Except for managing the import, export, and processing of content, it also handles various workflows of the system such as user management, access rights, local search and retrieval and it provides functionality for organizing content in groups and collections. Figure 2 describes the main DAM functionalities.

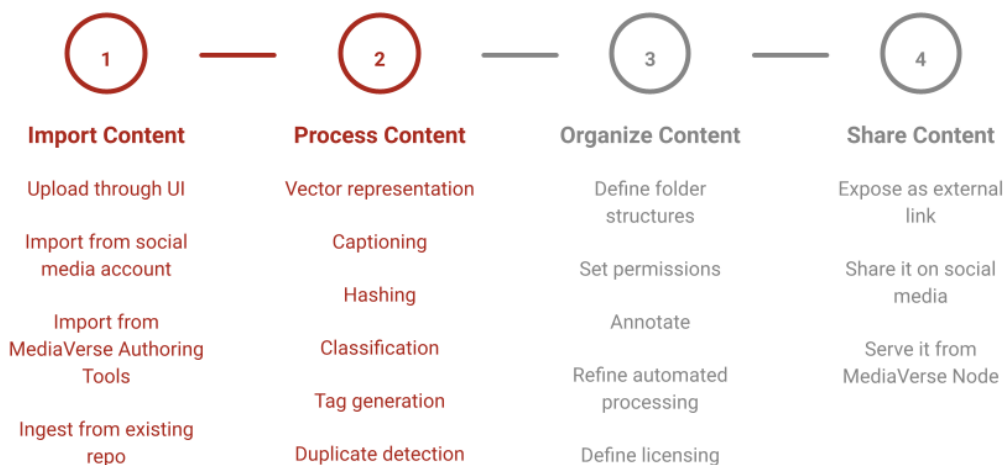


Figure 2: Main DAM functionalities

The functionalities of Figure 2 may be needed by other modules of the MediaVerse Node and are exposed by the DAM as consumable HTTP methods. This set of methods is continuously updated while the rest of the modules are progressing with development. We maintain a live version of the available methods through a Swagger interface⁴. A preview of the exposed API can be found in Figure 3.

The important architectural decision we reached regarding the role of the DAM inside the MediaVerse Node indicates that the DAM must control the MV Node storage layer and data-structures of all distributed (micro-service) elements of the Node. For instance, all IPR-related metadata and persistent objects are handled through the DAM. Therefore, CRUD actions towards the storage must pass through the DAM.

Except for exposing HTTP services to other MV Node components, the DAM also consumes the APIs of other MV Node elements and orchestrates several workflows as described at the beginning of this section. The way of consuming the available interfaces is explained in section 3.5.3 of Deliverable D2.2, while the high-level diagram of this interaction is shown in chapter 2.4.2 (Figure 16) of Deliverable D6.1.

⁴<https://mediaverse-node.herokuapp.com/swagger-ui/#/>

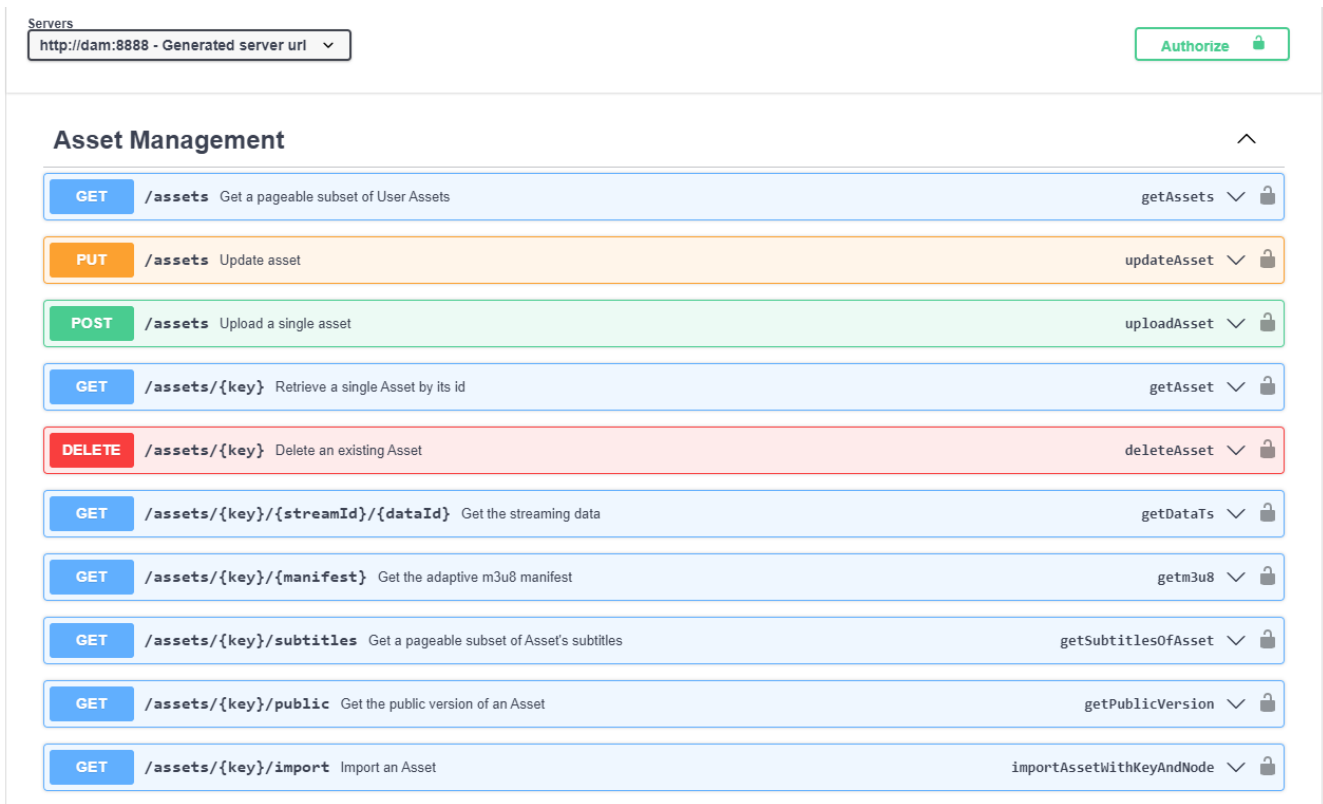


Figure 3: Swagger of DAM API

2.2 Technical Description

This sub-section describes the conceptual architecture, technical implementation and other technical aspects of the DAM.

2.2.1 Conceptual Architecture

The MediaVerse node implementation follows a service-oriented design. DAM is a monolithic application inside the node, it exposes a RESTful API and relates to the storage layer. The latter consists of a binary object storage, the searchable indexes, and metadata storage. Despite the fact that the storage layer can be physically deployed outside the implementation of the DAM (e.g., as cloud storage), conceptually it can be considered as part of the same application. Moreover, since all interactions to the storage layer are directed through the DAM, we can consider it as a standalone service, binding the following layers, as described in Figure 4.

This grouping is based on the role of each layer: The DAM API is responsible for the implementation of the exposed RESTful HTTP methods. The Business Logic implements all the algorithms and rules that are needed for the execution of the more complex tasks inside the application. The Domain Objects contain all the structures of the objects used inside the application. Finally, Data Access and Data Connectors are responsible for the communication (read, write, and update) with the Storage Media.

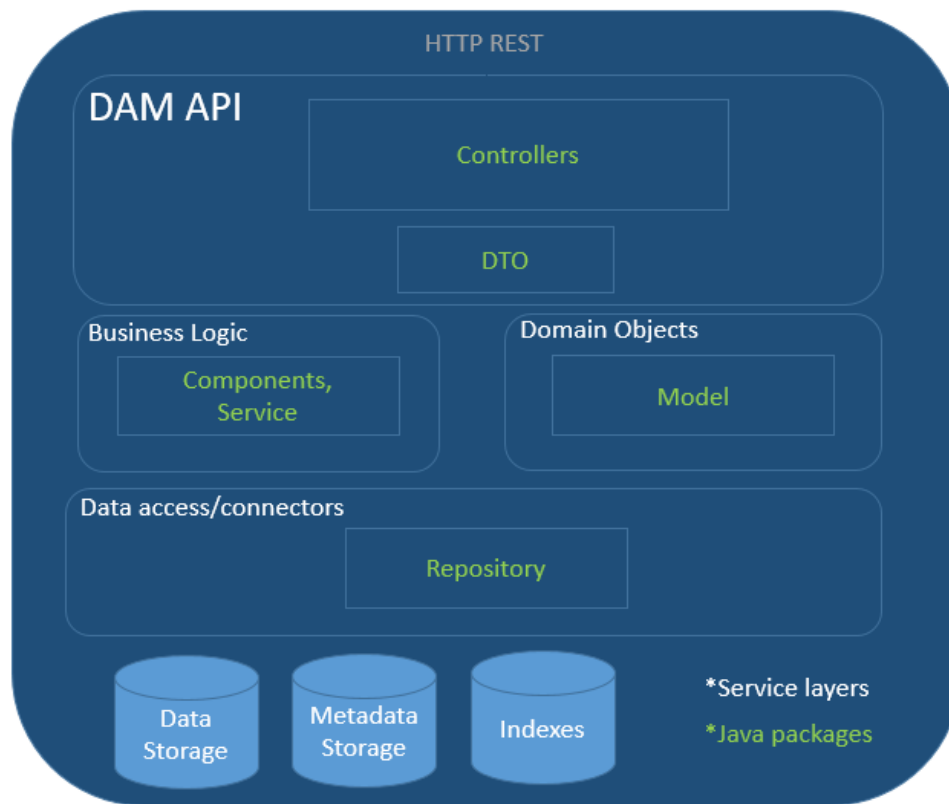


Figure 4: The DAM Service

2.2.2 Technical Implementation

All the DAM layers are implemented under a single software application in Java, which was selected because of its robustness and wide use in enterprise systems. More specifically, we used the Spring Boot framework⁵ with embedded Apache Tomcat⁶ as web server. The code structure in the internal GitLab repository is described as follows (and illustrated in Figure 5). The project is organized in Java packages (in parenthesis, we name the relevant conceptual elements in the layered architecture of Figure 4):

- **Controllers** (DAM API). This describes REST methods that handle external HTTP requests and messages.
- **Components** (Business Logic). This defines external components that can be used for communication with third party services, e.g., AWS S3 client.
- **DTO** (DAM API). This defines all the Data Transfer Objects used inside the message exchanges between the DAM and the external clients.
- **Model** (Domain Objects). This domain layer describes the structure of objects used in the business layer.
- **Service** (Business Logic). These services implement the business logic of the application.
- **Repository** (Data Access, Data Connectors). These are the connectors and CRUD logic with the Databases and other storage media of the ecosystem.
- **Exception**. These are the descriptions of all the exceptions that can occur inside the application.
- **Configurations, Enums, Filters, Utils**. These are internal implementations that include methods for supporting the above packages.

⁵<https://spring.io/projects/spring-boot>

⁶<https://tomcat.apache.org/>

Name	Last commit	Last update
..		
components	EUMV-80: Refactor BE codebase	7 months ago
configurations	Update versions	2 days ago
controllers	Merge branch 'feature/keeptrackofpublishedassets' into features/e...	2 days ago
dto	Add more boolean properties to BcData	1 month ago
enums	EUMV-160: Implement video sharing functionality for Twitter	1 month ago
exception	Add registration error when password does not meet the require...	3 days ago
filters	Export and import functionality first draft	1 week ago
model	Merge branch 'feature/keeptrackofpublishedassets' into features/e...	2 days ago
repository	Merge branch 'feature/keeptrackofpublishedassets' into features/e...	2 days ago
service	Added IPFS_URL as an env var	2 days ago
sortRepositories	EUMV-34: Complete refactor to submodules	1 year ago
utils	Filter only registered Assets	2 days ago
MediaverseApplication.java	Add filtering and sorting for getting assets	3 months ago

Figure 5: Code structure

The **Storage Systems** used for the persistence of the various objects (metadata, binary files, indexes) are currently defined as follows:

- MongoDB as NoSQL Database for the metadata of the assets and other entities stored in a MV node
- Apache Solr for the full text search and indexing of assets
- IPFS for the inter-node communication, including federated search and SLC storage
- Local Volume, AWS S3, Azure or other for the binary storage of assets. This is agnostic by design and can be handled programmatically through Data Connectors implementing the same interface. Following the adapter design pattern, the DAM could switch to support any storage implementation needed.

The above packages have been updated to provide the below functionalities:

1. Local and external authentication. Both local authentications, the process of authenticating a MediaVerse user, and external authentication, the process of authenticating a user from a 3rd party application such as Fader and VRodos are implemented.
2. Uploading of media files. DAM supports the uploading of Image, Video, Audio, Text, 3D, and Subtitle media files.
3. Publishing to social media platforms. Videos can be uploaded to Twitter and YouTube (the latter is still not included in the UI); Text and Images can be uploaded to Twitter directly from the MediaVerse dashboard; in addition, an application is currently reviewed by TikTok.
4. Searching capabilities. The searching functionality has been updated to support filtering of assets by media annotation results. Searching through external sources such as Wikimedia has been added as well.

5. Triggering transcoding services and storing the results. HLS streaming and preview files are stored with a reference to the original file.
6. Original/Preview Files Retrieval and Importing from one node to other.
7. Support of HTTP requests from the IPR service which handles all the blockchain transactions.
8. Moderation Rules operations.

2.2.3 Secure Communication

In terms of security, the DAM is using server signed JWT tokens, which are validated on every request to the DAM by external clients (e.g., browser or external applications). The formulation of such JWT tokens takes place during the OAuth 2.0 handshake and uses a private passphrase to protect it from replication. More specifically, once the user is authenticated with the use of credentials, the DAM creates a JWT token, signed by itself with a passphrase. This means that only the DAM server that owns the passphrase can validate it and no other component is able to create valid tokens. Once a token is returned to the client web app, it is stored to the browser's local storage to be used in all subsequent requests.

A significant advantage of the DAM application is that it is loosely coupled with the front-end interface. This means that there is no maintained open session between the triggering part and the DAM backend. As explained, every call is stateless and authenticates the actor every time with the same JWT token. This also facilitates the replication of the backend in multiple serving instances, offering load balancing, high availability, rolling upgrade functionality, etc.

3 Dashboard – UI

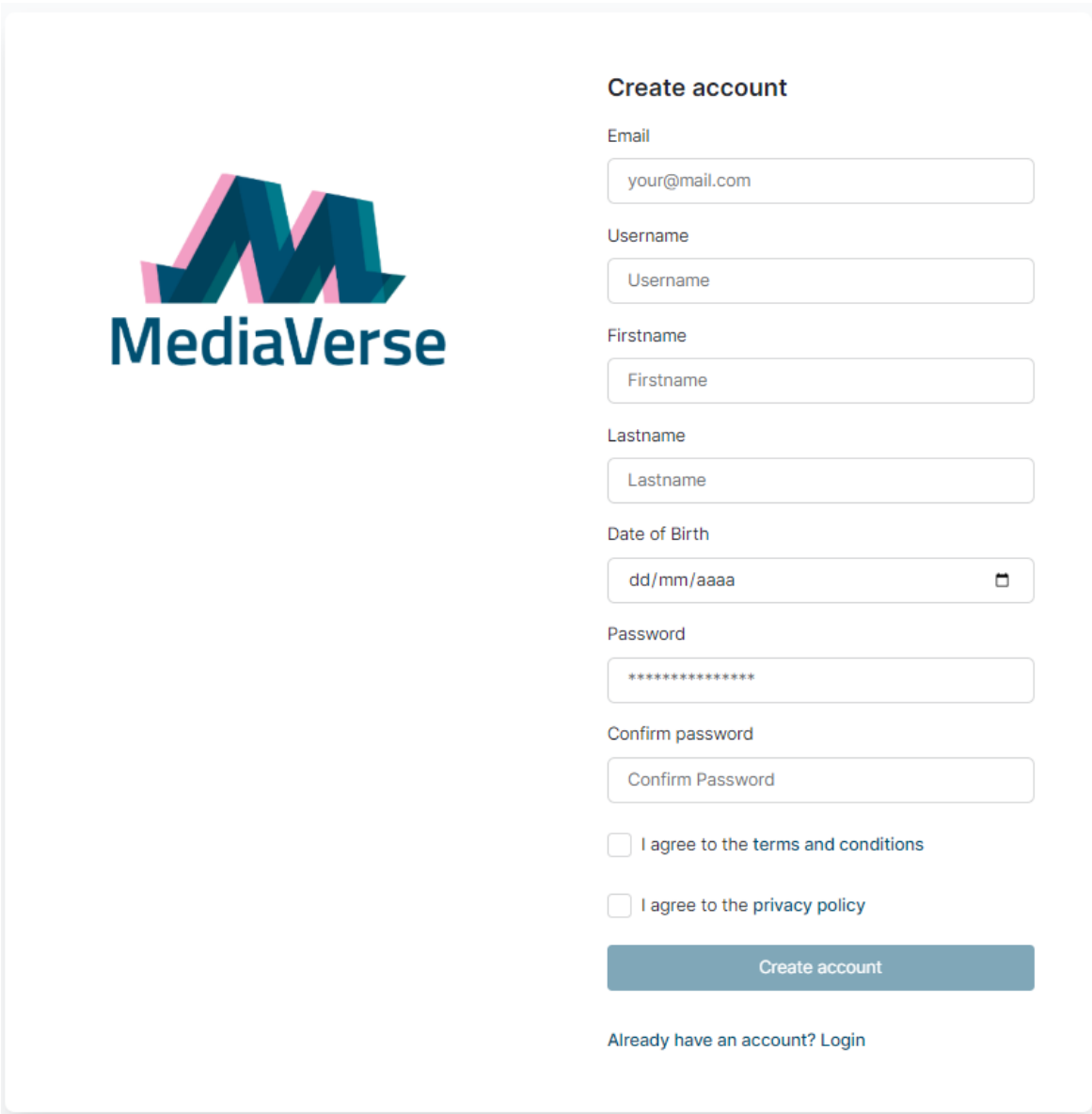
This section is about the MediaVerse User Interface (UI). The UI is designed to support the management of audio, video, images, text, and 3D models assets.

3.1 Functional Description

The UI is integrated with the DAM and other services inside each MV node, including the IPFS, and IPR. External tools will be integrated to the UI in the future (e.g., RACU or Fader).

3.1.1 Login

The login is the first contact of the user with MediaVerse: if the user has no MV login, the user needs to click on “Create Account” that leads to the registration form (see Figure 6).



The image shows a 'Create account' form for MediaVerse. On the left is the MediaVerse logo, which consists of a stylized 'M' made of blue and pink geometric shapes above the word 'MediaVerse' in a bold, dark blue sans-serif font. The form itself is titled 'Create account' in bold. It contains several input fields: 'Email' (with placeholder 'your@mail.com'), 'Username' (with placeholder 'Username'), 'Firstname' (with placeholder 'Firstname'), 'Lastname' (with placeholder 'Lastname'), 'Date of Birth' (with placeholder 'dd/mm/yyyy' and a calendar icon), 'Password' (with placeholder '*****'), and 'Confirm password' (with placeholder 'Confirm Password'). Below these fields are two checkboxes: 'I agree to the terms and conditions' and 'I agree to the privacy policy'. At the bottom of the form is a blue button labeled 'Create account' and a link that says 'Already have an account? Login'.

Figure 6: Create Account Window

In this window, the user needs to complete the next fields in order to obtain a MV login:

- Email
- Username
- First Name
- Last Name
- Date of Birth
- Password (2-Step)

First, the user must read and accept “Terms and Conditions” and “Privacy Policy” to create a user login. Once the account is created, the user can login to MediaVerse dashboard (Figure 7).

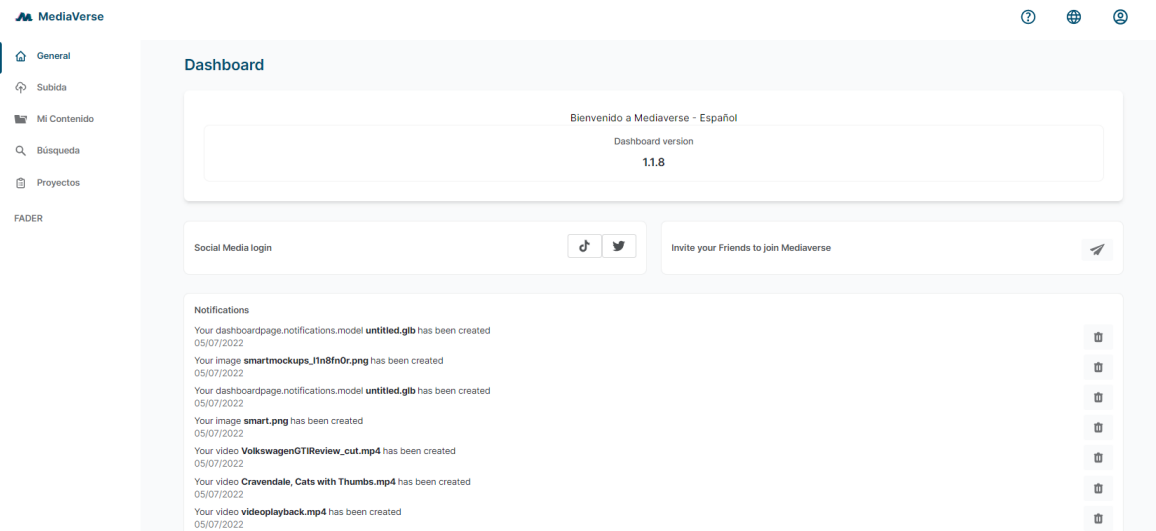


Figure 7: MV Default Tab (Dashboard)

3.1.2 Dashboard

The dashboard (Figure 7) displays user information related to his/her previous activities with media assets. There is a section for social media login, another to invite collaborators to MV, and a last section for user notifications.

Social Media

Figure 8 illustrates the social media login. Currently, there are buttons for TikTok and Twitter. There are plans to add YouTube which is already available in the backend.

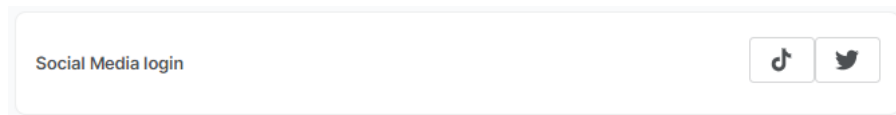


Figure 8: Social Media Login Section

TikTok: Right now, the button redirects to the TikTok official webpage. If the MediaVerse application is approved by TikTok, the UI will directly support integration with TikTok via the social login.

Twitter: When the user clicks the button for login, MediaVerse redirects him to a window (Figure 9) in order to insert his/her Twitter⁷ credentials and to link the associated Twitter account to the MV session.

⁷ Twitter: <https://twitter.com/>

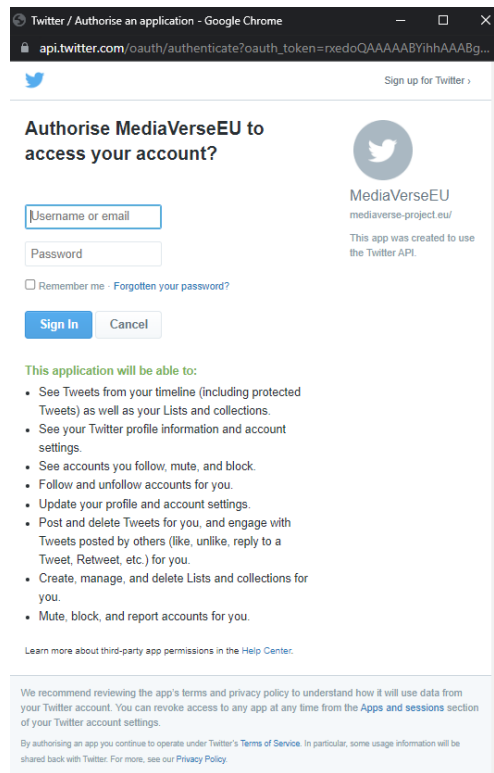


Figure 9: Twitter redirection to Login

Invite people to MediaVerse

This gives the possibility to invite an external person to MediaVerse. When the user clicks on the button an email window (default email application) appears to send the invitation. In the future this functionality will improve.

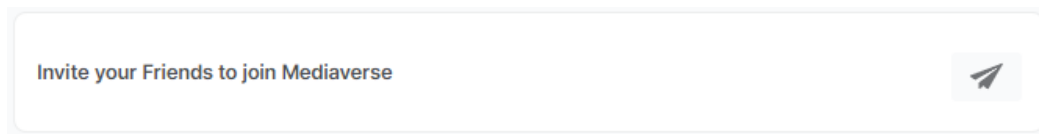


Figure 10: Invite to MV card

Notifications

This functionality is under development. In the future more types of notifications will be supported. Now notifications only add the relative messages when an asset is uploaded (see Figure 11).

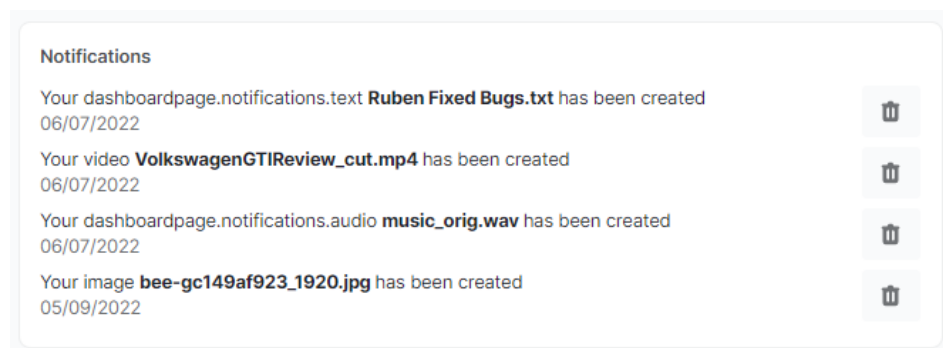


Figure 11: Notification Card

3.1.3 Profile

In the top right bar, the button to the right leads to the user profile (Figure 12).

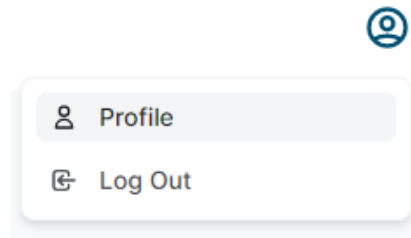


Figure 12: Profile (Top Bar)

Users can find information related to their user account (Figure 13), billing information (Figure 14), subtitle preferences, user role in MV, etc. The implementation of this feature is ongoing.

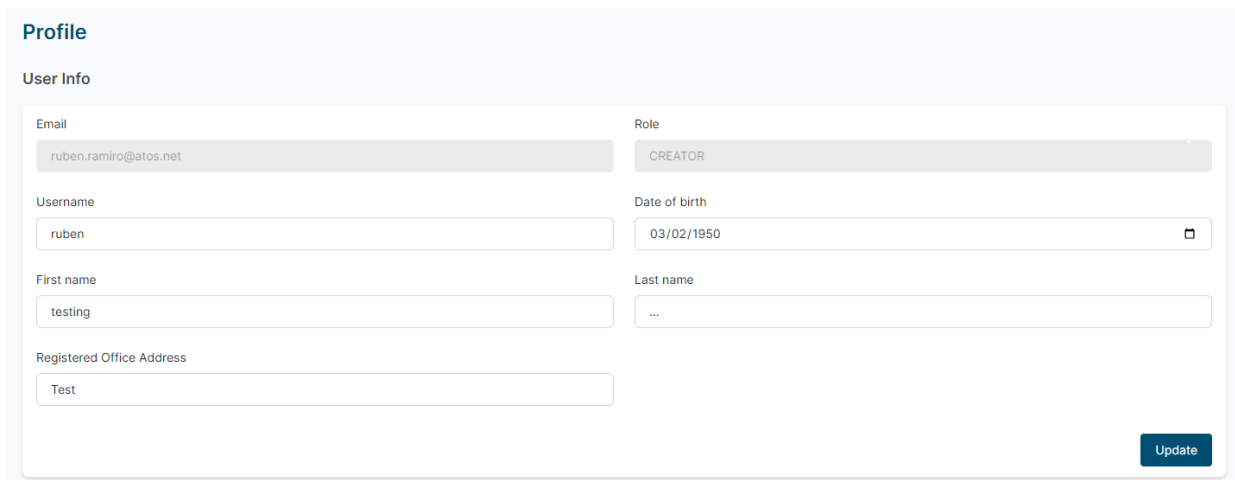
A 'Profile' section titled 'User Info' containing a form with several input fields. The fields are: Email (ruben.ramiro@atos.net), Role (CREATOR), Username (ruben), Date of birth (03/02/1950), First name (testing), Last name (...), and Registered Office Address (Test). An 'Update' button is located at the bottom right of the form.

Figure 13: User info

In Figure 14 the user can see the Billing information/Wallet part that contains the blockchain of the user and the preferred fiat currency. The user can choose among EUR, GBP, USD, and CNY.

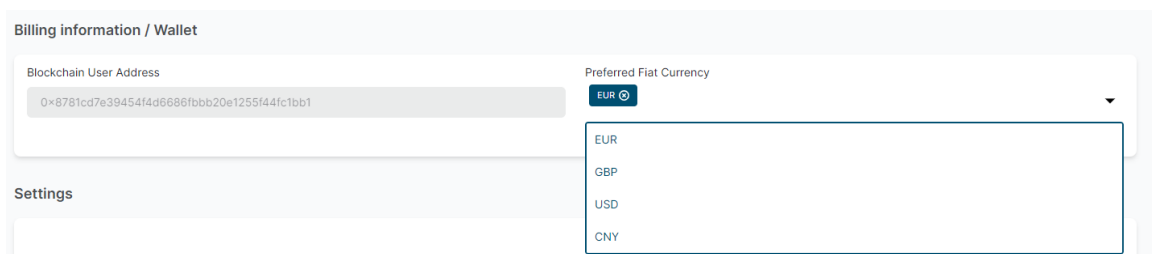
A 'Billing information / Wallet' section containing a form. It has two main parts: 'Blockchain User Address' with a text field showing a long alphanumeric string, and 'Preferred Fiat Currency' with a dropdown menu. The dropdown menu is open, showing options: EUR, GBP, USD, and CNY. Below these is a 'Settings' section which is currently empty.

Figure 14: Billing Information / Wallet

3.1.4 Language

The default language that the MV UI applies is the default language in the user browser. The user can manually select a different language (as for now, English, Catalan and Spanish are available, as shown in Figure 15).

When the user changes the language, the changes can be seen in different MV sections: in the left bar, in the user profile and in some parts of the dashboard. In the future, MediaVerse will extend into other sections the translations and will add more languages.

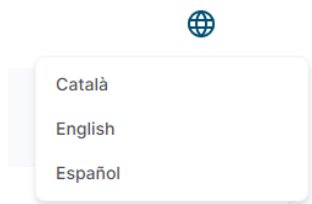


Figure 15: Languages Available

3.1.5 Help

This functionality offers the user a brief step-by-step tutorial on the structure of the UI, as well as the functionality of each section of the left bar (Figure 16).

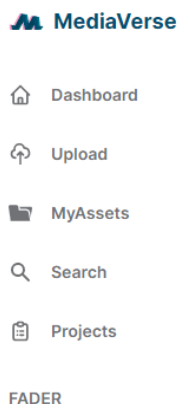


Figure 16: Left Bar

This functionality is work in progress, the intention is to extend it for other sections of the UI. To activate it, the user needs to click on the “Help button”. When the user activates it, the UI will present a series of snippets explaining the role of each section in MV (see Figure 17 and Figure 18).

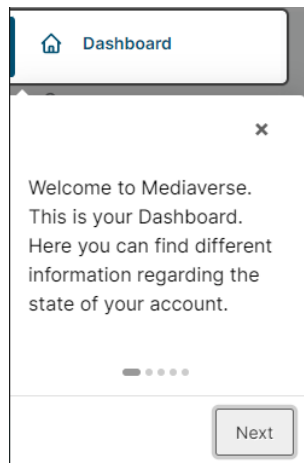


Figure 17: Dashboard Section Description

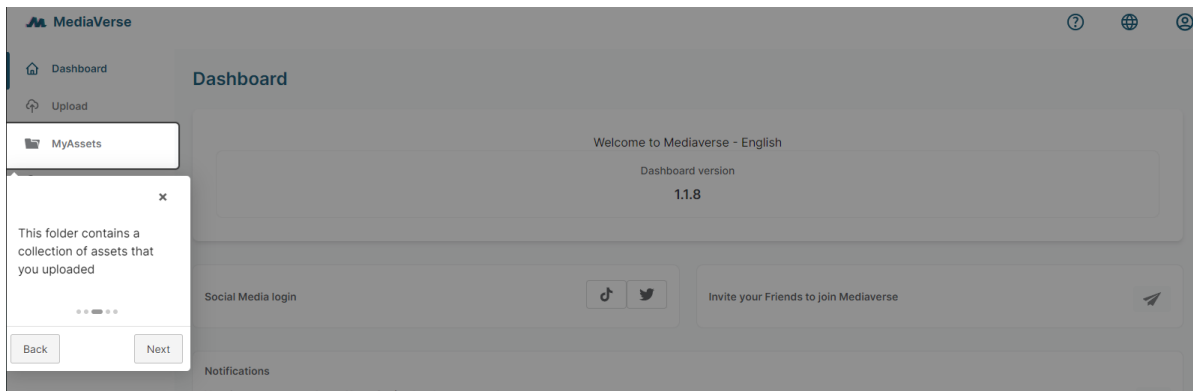


Figure 18: Help Functionality

3.1.6 Upload

In this section, the MediaVerse node allows the user to upload multiple asset types: audio, videos, images, texts, and 3D Models. There are also Content Guidelines to warn the user about what type of content is allowed on MV platform (Figure 19). The allowed formats on MV:

- Audio: wav, mpeg, mpeg3, ogg and flac
- Video: QuickTime and mp4
- Images: jpeg, png and jpg
- Text: json and text/plain
- 3D models: gltf-binary and obj

Upload

Please, upload your content here by dropping it in the box or by tapping on it.

Right now only pictures and video are accepted as content.

Drop your file here

Content Guidelines

You may not submit any content that:

- Infringes any third party's copyrights or other rights (e.g., trademark, privacy rights, etc.)
- Is sexually explicit or promotes a sexual service;
- Is defamatory;
- Is harassing or abusive;
- Contains hateful or discriminatory speech;
- Promotes or supports terror or hate groups;
- Contains instructions on how to assemble explosive/incendiary devices or homemade/improvised firearms;
- Exploits or endangers minors;
- Depicts or encourages self-harm or suicide;
- Depicts (1) unlawful real-world acts of extreme violence, (2) vivid, realistic, or particularly graphic acts of violence and brutality, (3) sexualized violence, including rape, torture, abuse, and humiliation, or (4) animal cruelty or extreme violence towards animals;
- Promotes fraudulent or dubious money-making schemes, proposes an unlawful transaction, or uses deceptive marketing practices;
- Contains false or misleading claims about (1) vaccination safety, or (2) health-related information that has a serious potential to cause public harm;
- Contains false or misleading information about voting;
- Contains (1) claims that a real-world tragedy did not occur; (2) false claims that a violent crime or catastrophe has occurred; or (3) false or misleading information (including fake news, deepfakes, propaganda, or unproven or debunked conspiracy theories) that creates a serious risk of material harm to a person, group, or the general public; or violates any applicable law.

This example of content guidelines was taken from [Vimeo](#).

Figure 19: Upload Section

There are two ways for the user to upload an asset: dragging into the interactive upload zone or clicking on it. If the user clicks in the interactive upload zone, the file explorer will open a dialog to choose and upload an asset. Once the asset is uploaded in the right format, the UI shows to the user a success message (Figure 20).

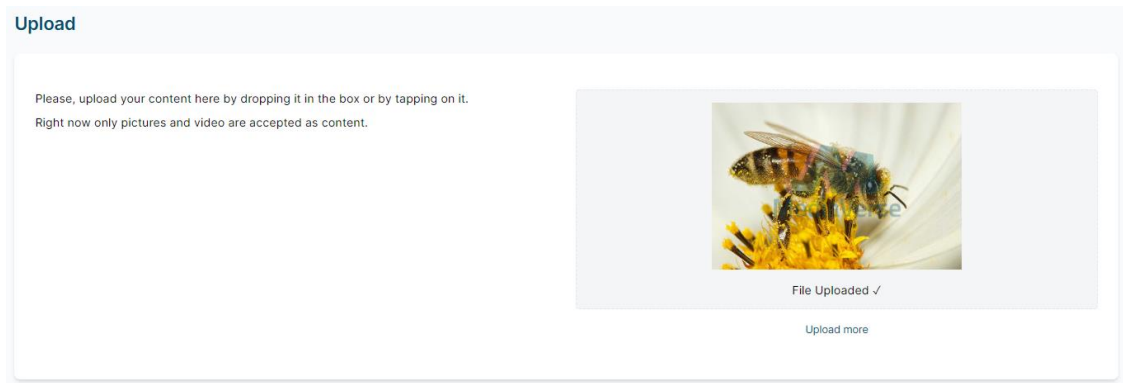


Figure 20: Successfully Upload

In case of failure, an appropriate notification message is shown (see Figure 21):

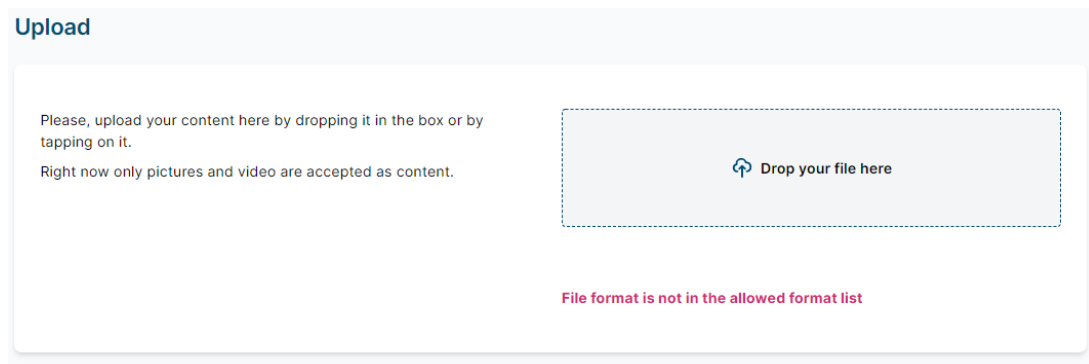


Figure 21: Failed Upload

3.1.7 MyAssets

The first contact with assets once inside the “MyAssets” section will be through the previews of the assets that belong to the user (Figure 22). For videos, a gif preview is used. Assets can be filtered using the Filter Button in the top right corner position (Figure 23). When the user clicks on the button, a new dialogue pops up for the user to control the filters.

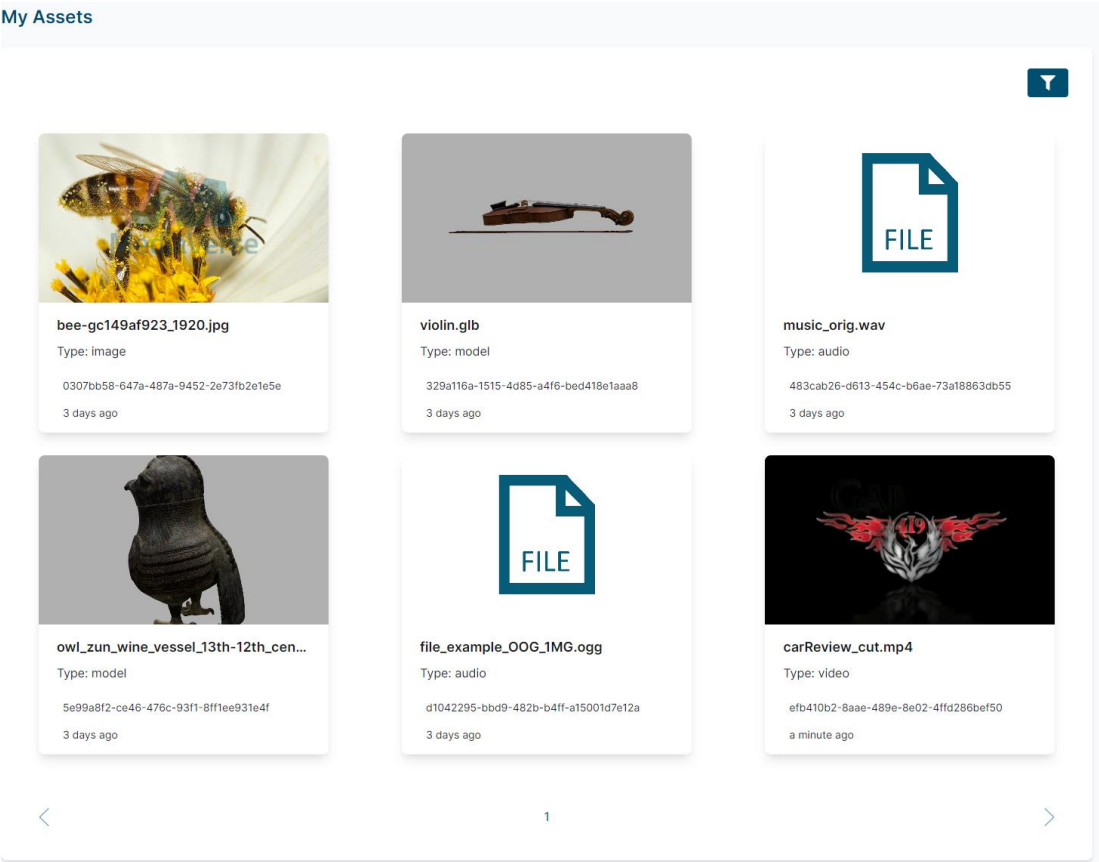


Figure 22: My Assets view

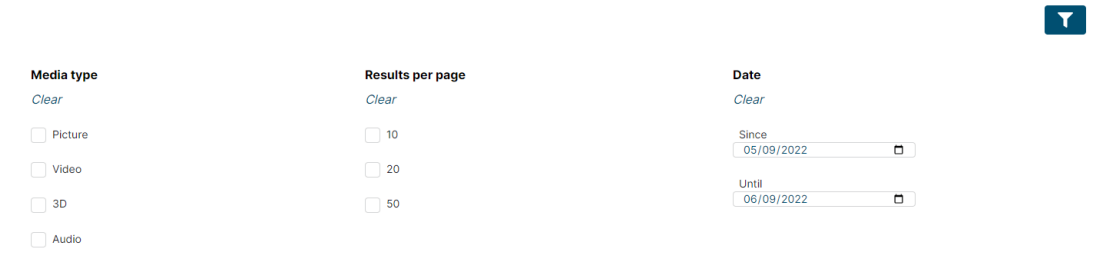


Figure 23: Filter Controls

3D Models

The 3D model is represented in MV via the JS library Three.js⁸. The user can interact with the model, zooming out, zooming in, and moving the focus around the object (Figure 24).

⁸ Three.js: <https://threejs.org/>

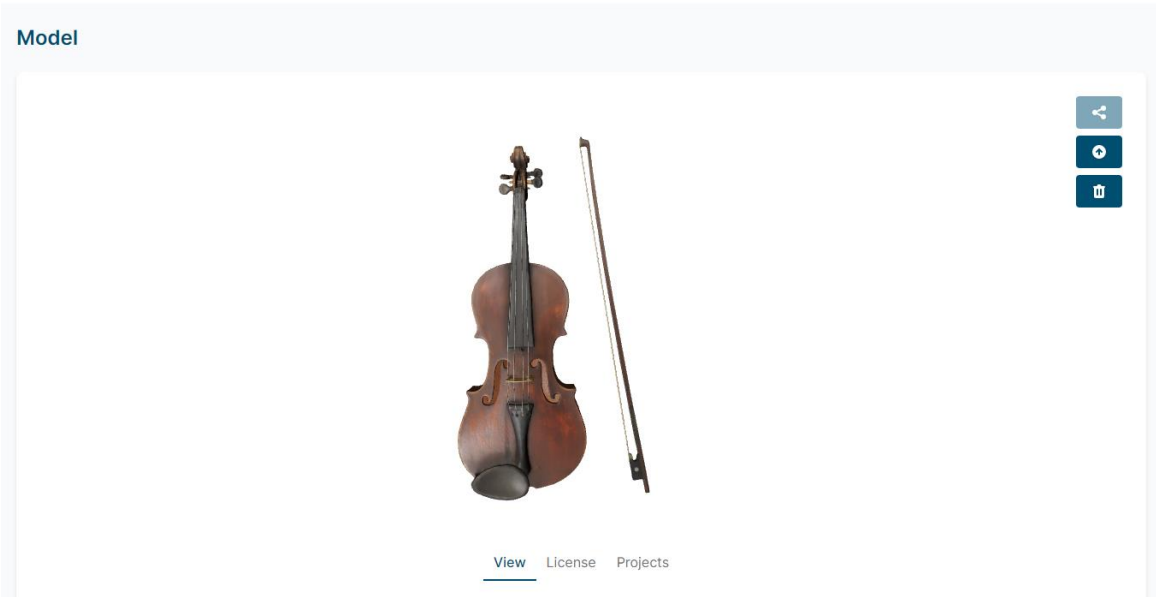


Figure 24: Three.js for 3D models

Video

After the user uploads a video, in the background the DAM creates the associated subtitles in multiple languages (English, Russian, Italian, Spanish and German). Once the subtitles are ready, they are automatically attached to the video when the user is playing it in the MV player (Figure 25).

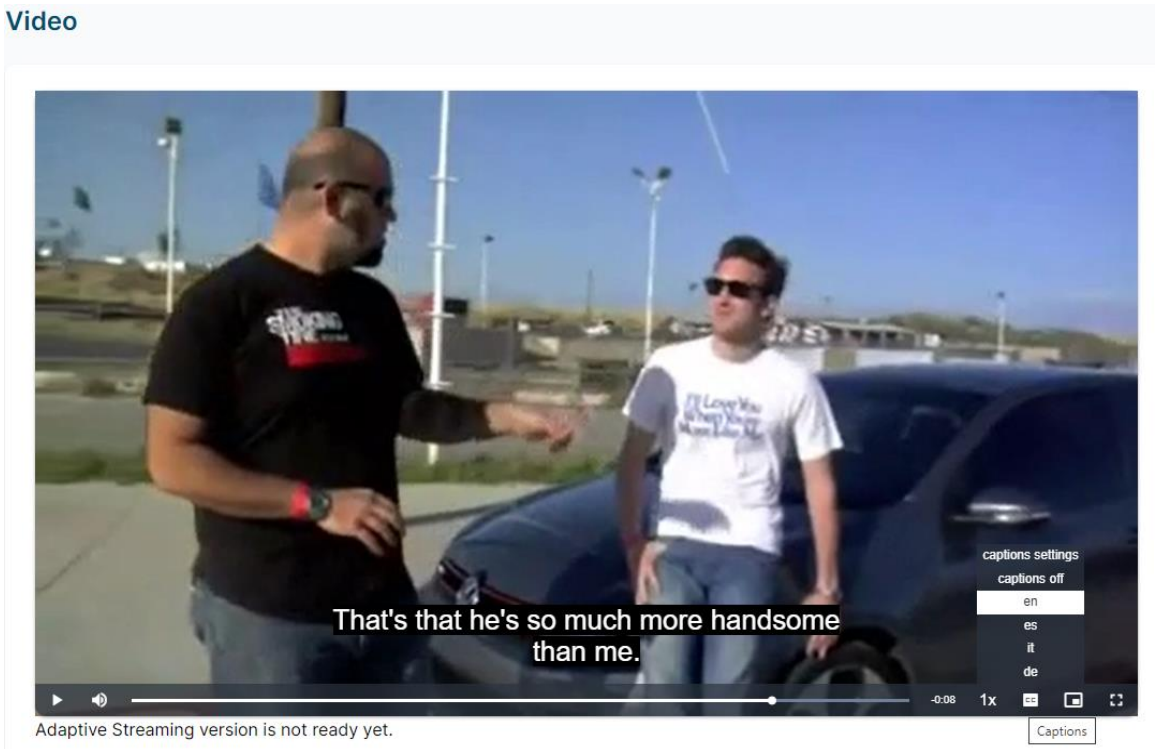


Figure 25: Captions for video

Disturbing Content Moderation

For images and videos that contain disturbing content, MediaVerse offers support for moderation. This moderation system is automated through AI as detailed in deliverable D5.4 - Content Moderation Toolset, but moderation can be also done manually using labels on each asset. The result in both cases is the addition of an appropriate warning message at the asset preview (Figure 26).

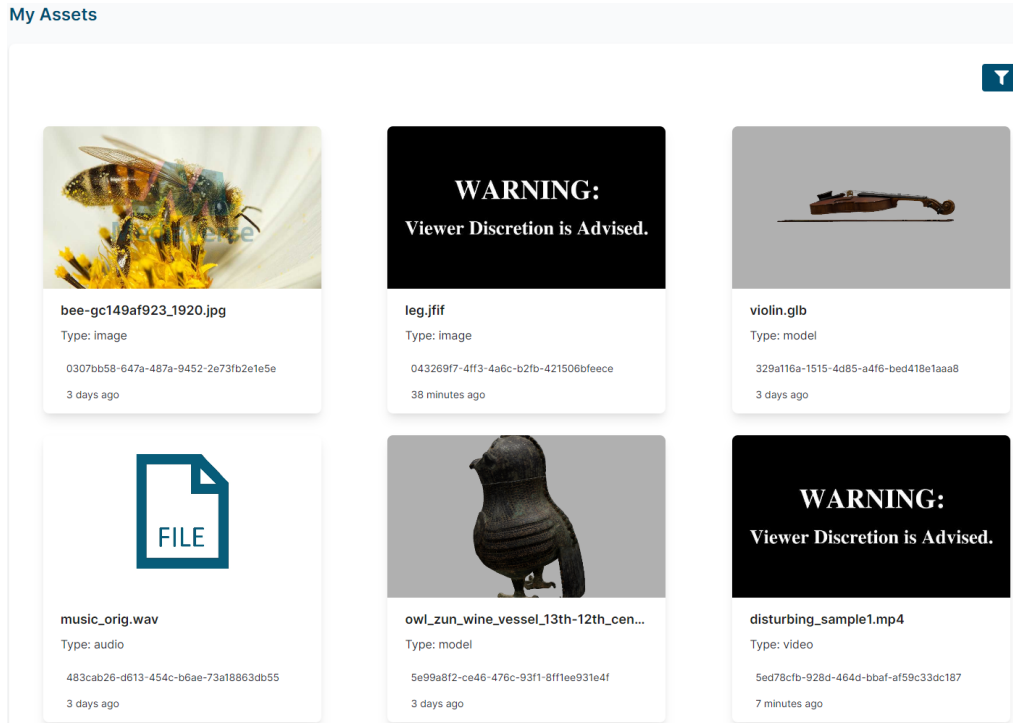


Figure 26: Disturbing Content Moderation Example

Share via Twitter

Inside the asset card, the user could find the button to share the asset via social media. To share the asset, the user needs to be logged on Twitter. The Share Button is on the top right corner (Figure 28). Once the Share button is pressed, a dialog is shown (Figure 27). After pressing the button “Post” a message will appear at the top of the window to confirm or not the success of the Twitter publication.

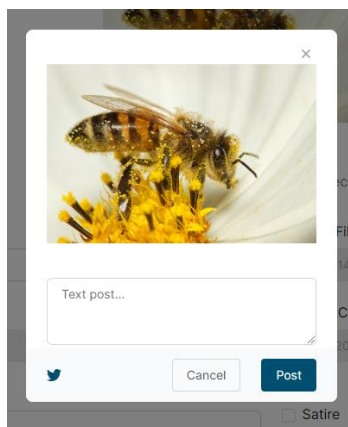



Figure 27: Share via Twitter

Metadata View

If the user presses one of the assets cards the user can view the asset metadata section (Figure 28).

Image



[View](#) [License](#) [Projects](#)

Description	Original Filename
bee-gc149af923_1920.jpg	bee-gc149af923_1920.jpg
Content Type	Content Creation Date
image/jpeg	02/09/2022
Price	Flags
cost	<input type="checkbox"/> Satire
Available Languages	Labels
<input type="text" value="Add..."/>	<input type="text" value="Add..."/>

[Update](#)

Automatic Annotations

Image Action Recognition	Image Captioning
brush_painting	not found
Image Disturbing Content	Image Face Recognition
no	not found
Image Object Detection	
not found	

Figure 28: View of an asset

In this window, the user can view and check all the metadata related to the selected asset. For now, the fields include the following:

- Description
- Original Filename
- Content Type
- Content Creation Date
- Price
- Flags (e.g. Satire)
- Available Languages
- Labels based on automatic annotations⁹, including Image Action Recognition, Captioning, Disturbing Content, Face Recognition, and Object Detection

⁹ D3.2 - Content Discovery and Recommendation, Annotation and Adaptation Framework: <https://nxcloud.mediaverse-h2020.eu/s/wtPEFKgzz9QcY8j>

Adj...

EN
CA
ES
FR
DE

Figure 29: Languages


Adj...

NSFW
HATE
VIOLENT
FAKE

Figure 30: Labels to add manually moderation to the asset

License

Section “License” (Figure 31) is related to the registration actions using the Creative Commons Licenses¹⁰ context.



ViewLicenseProjects

Select a License ^o

No Set

Confirm

Do you need help choosing a license? ☒

Figure 31: License Tab

If additional help is needed to choose the best CC license, the user can use the license advisor (Figure 32).

License Advisor

Follow the steps to select the best license for your content.

Do you want attribution for your work?

Anyone can use my work, even without giving me attribution.

Anyone using my work must include proper attribution.

Figure 32: License Advisor

¹⁰ Creative Commons Licenses: <https://creativecommons.org/licenses/?lang=en> EN

Once the user has selected a license, the system returns the status of the registration (Figure 33 and Figure 34). It is mandatory to have the “Registered Office Address” field completed in the profile before registration.

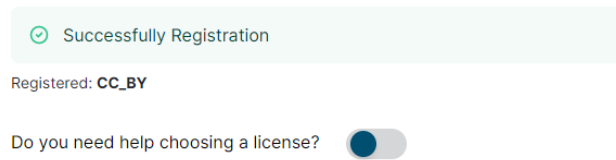


Figure 33: Successful CC license registration



Figure 34: Failed registration

Projects Tab inside Asset View

A MediaVerse project is a group of assets and associated users. A project can be associated with multiple MV users and multiple assets. This section illustrates the project tab (Figure 35), which will be explained later in section 3.1.9.

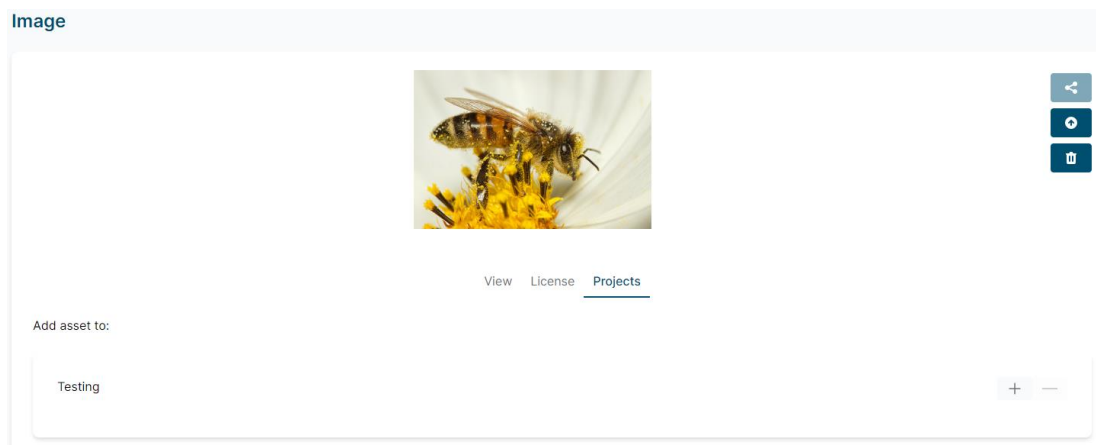


Figure 35: Projects Tab

3.1.8 Local and Federated Search

The user has two options to search for assets in the MV network:

- Local Node Search: this search is limited to the local MV node, i.e. the node that triggered the search.
- External Node Search (Federated¹¹): this search extends to also the other nodes in MediaVerse.

¹¹ Further details on Federated search are provided in section 4.2.1.

Local Node Search

Once the user types the query into the local search, the local search service will search for all the assets associated with the query inside the local node. It will retrieve them and show them in the UI (Figure 36).

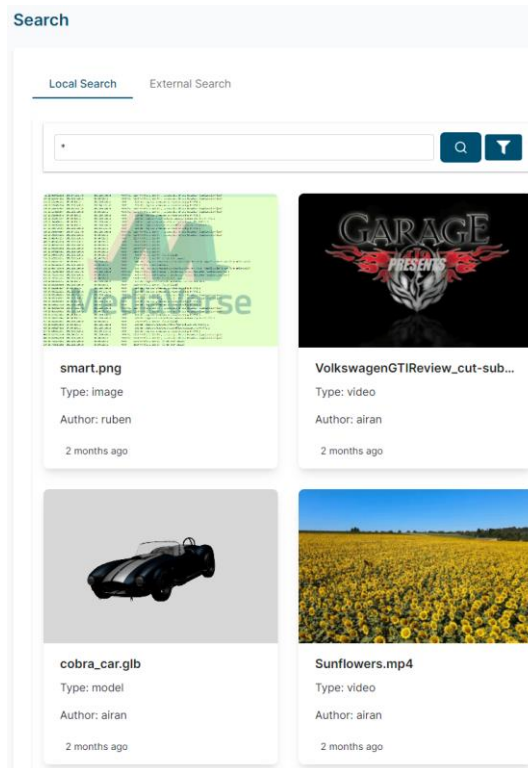


Figure 36: Local Search

External Node Search (Federated Search)

For this functionality, the UI uses IPFS¹² to extend the search to the other nodes in the MV network. After the search, the UI presents all the retrieved assets from each node (Figure 37 and Figure 38).

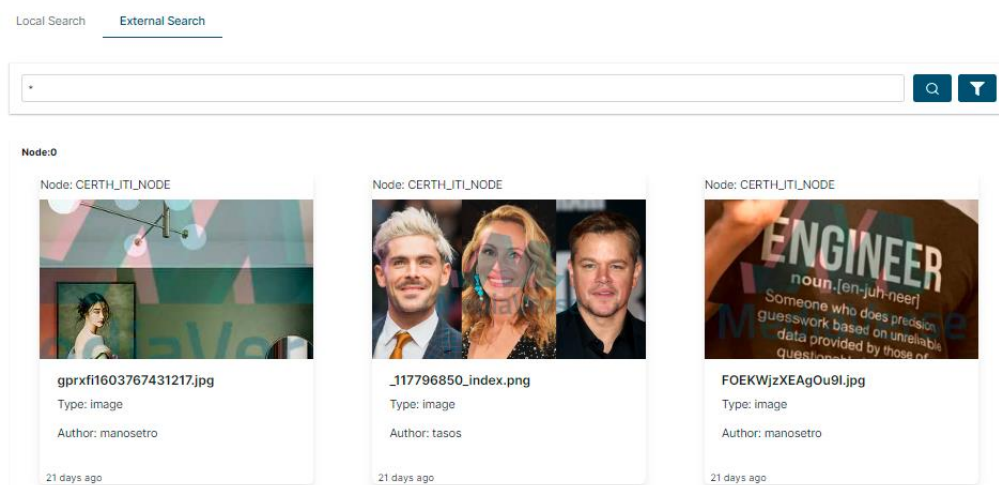


Figure 37: External Search, CERTH node

¹² Take a look at pubsub on IPFS. (2022). Retrieved 11 August 2022, from <https://blog.ipfs.io/25-pubsub/>

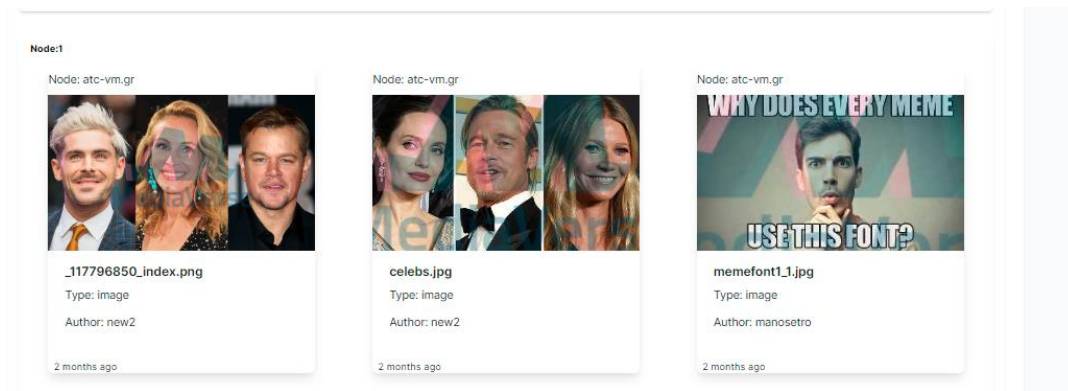


Figure 38: External Search, ATC node

The external federated search will be improved offering the user a better experience, e.g., sorting assets by relevance (number of views, number of purchases, etc.) instead of sorting by nodes.

3.1.9 Projects

MediaVerse allows users to organize their assets in projects. Inside a project, users can add assets that are already stored in MV. In the future, other users could be added to produce collaborative content.

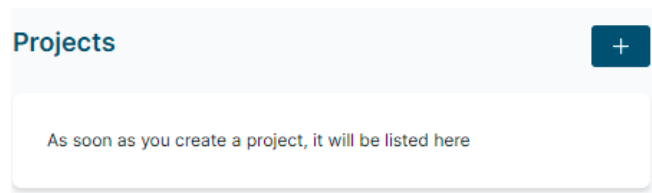


Figure 39: Project description.

The Project section is composed of a Create New Project button located on the right-hand side of the viewport (Figure 40) and a Project List located at the centre of the viewport.

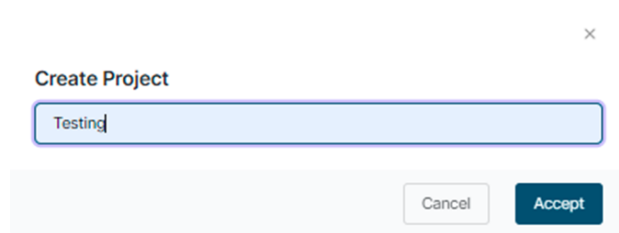


Figure 40: Create New Project.

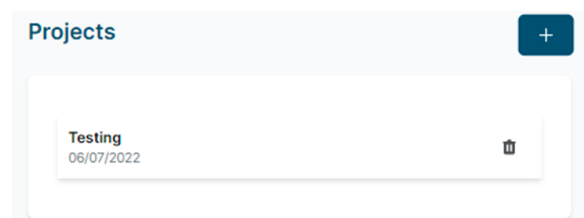


Figure 41: Project created

To create a new project, the user needs to click on the Create New Project button. A popup window will prompt the user for a name for this new project and after filling out the information and clicking on Accept to confirm, a new project is created. The newly created project will appear as a part of the Project List. Every project created by the user will be stored in the Project List and can be deleted if needed, by clicking on the “Trashcan Icon” in the right-hand side of the viewport (Figure 41). A popup window will warn the user and will ask for confirmation before deleting.

Furthermore, by clicking on the name of the project, the user can navigate into the inner Project Details. A project is composed of the following parts, Information, Project Result, Assets and Users.

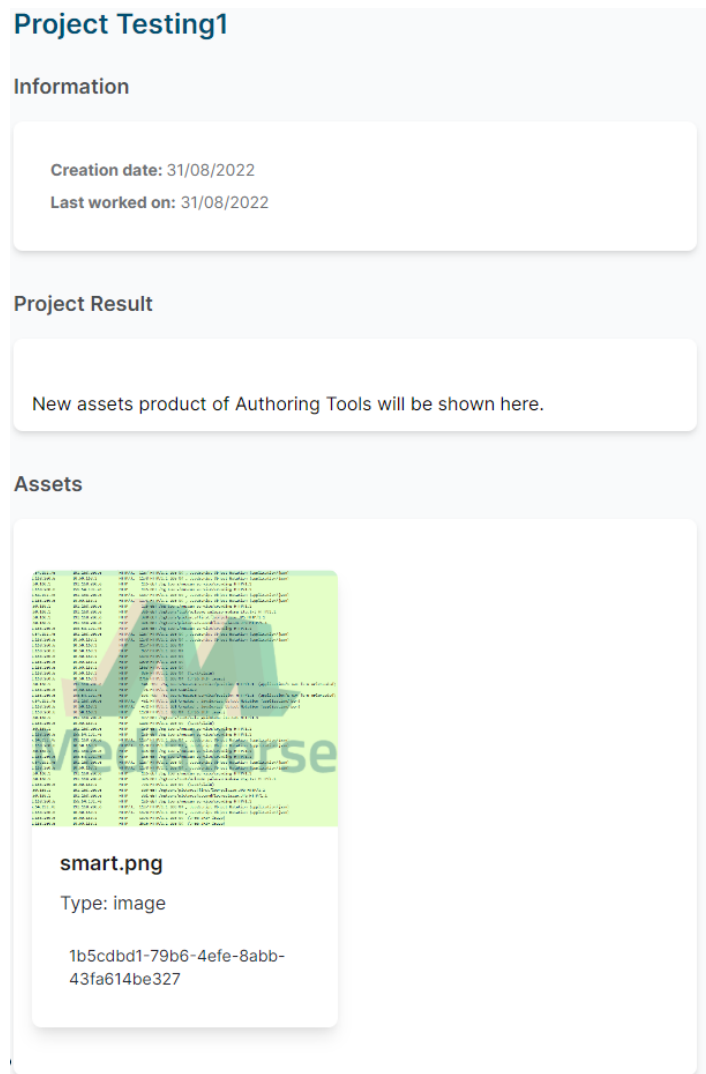


Figure 42: Project Details Description

The information section shows general information about the project. At the time of writing this document, it stores the project creation date and a few additional metadata. The Project Result will store the final result of the project, the new asset that will be the user’s production and information about it (Figure 42).

The Assets section lists assets that were added to the project by the project owner. These will be used by the user to create new assets. If an asset is added to the project, an asset card will display the asset in this section. The project owner can click on the asset card and navigate to the Asset view and remove the asset from the project on demand (Figure 43).

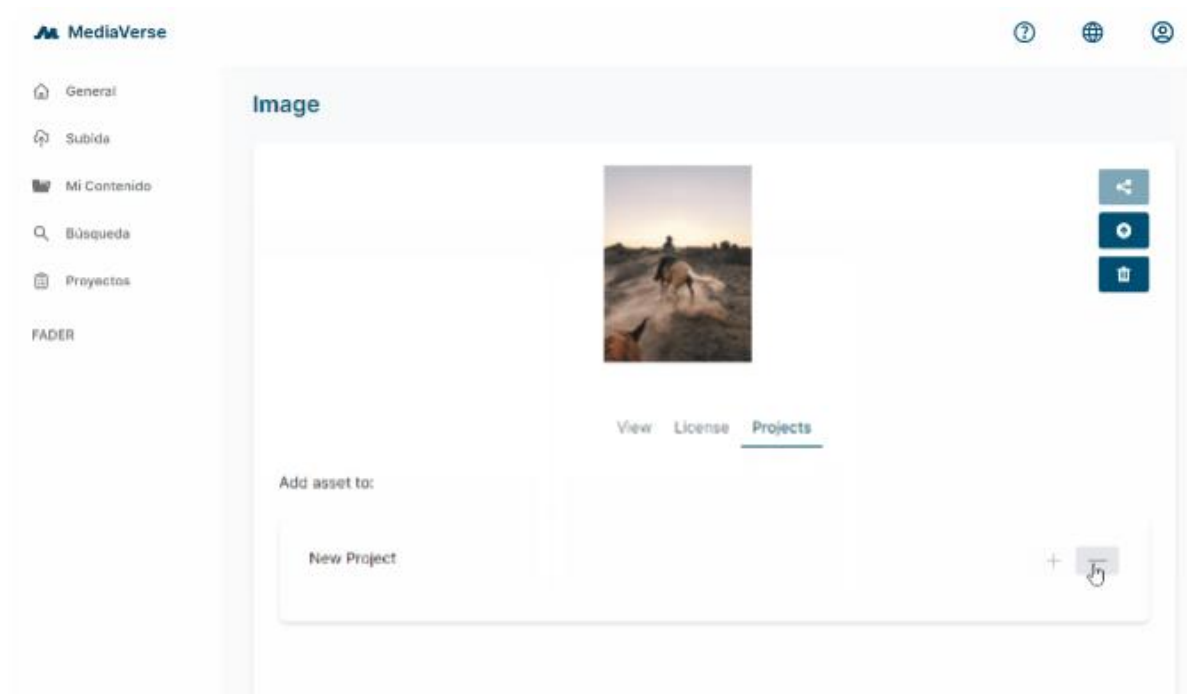


Figure 43: Project assets

The Users section contains a list of users, added by the project owner, to collaborate in the production of the new asset. A project owner can add or remove users to the project. Using the search function and typing a few characters, the project owner can search for collaborators. When the project owner has found the user they are looking for, they can add him/her to the user list located at the top of the search bar. A project owner can easily delete any user from this list by clicking on the Cross Icon at the right of each user (Figure 44).

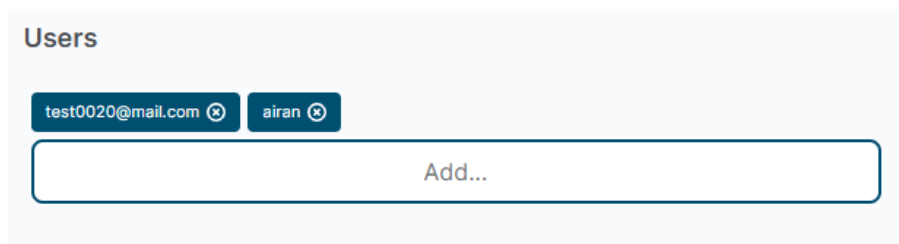


Figure 44: Adding and removing users from the project.

3.2 Technical Description

The UI is a front-end web application made in ReactJS¹³, this is an open-source JavaScript library used for building single-page applications that can dynamically adapt their content without reloading the page. In addition, the following libraries are used:

- **Tailwind CSS**¹⁴: a utility-first CSS framework for rapidly building custom user interfaces.
- **React Three Fiber**¹⁵: a React renderer for three.js. The UI is used for previewing 3D content.

¹³ <https://reactjs.org/>

¹⁴ <https://tailwindcss.com/>

¹⁵ <https://github.com/pmndrs/react-three-fiber>

- **Video.js**¹⁶: a web video player. It supports numerous video formats, including adaptive streaming formats, with a focus on accessibility and it's compatible with 360 videos.

The dashboard UI code is structured in the following folders:

- **Assets**: It contains all the assets used in the app. Most of the content is images or icons.
- **Components**: The code has been structured in components, that are independent blocks representing sections of the web app and that can be easily moved or reused in other sections.
- **Containers**: Containers are special components that add common functionality to UI components or provide the data.
- **Context**: react contexts give a way to pass data through the component hierarchy without having to pass props down manually.
- **Firebase**: this library is used to provide authentication for Twitter services and in the future for YouTube.
- **Helpers**: general tools to develop healthily code.
- **Pages**: Contains the different sections of the web app. Every page has its route so it can be accessed directly from a URL.
- **Routes**: Routes link specific paths with pages, making sure that when a URL is introduced the app will render the appropriate section requested by the user.
- **Store**: A store is a state container that holds the application state that will be shared between all the components that need it.
- **Styles**: This folder contains all the CSS files needed to style the application.
- **Translations**: This folder has a subfolder per language supported by the application. Every text in the app that needs to be translated will need to be declared here.

¹⁶ <https://videojs.com/>

4 Decentralized Framework for MediaVerse Communication

4.1 Introduction

This section reports on the progress made towards the development of the MediaVerse decentralized framework for communication and content exchange. The main goal has been to develop the interface for communication among MediaVerse nodes with the following requirements:

- Enable the discovery of reachable MV nodes, assuming that a node in a decentralized network may be unaware of the existence of the rest of nodes.
- Enable the search of content across the MV network, with each MV node holding its own Solr index.
- Enable the agile right management and copyright negotiation across the MV network.
- Enable the exchange, retrieval and sharing of digital assets between MV nodes.

As described in D6.1, a group of services has been created to meet these requirements. A first proof of concept was developed in the form of micro-services, and certain decisions were made about the protocols to be used. Starting from those decisions, we created the first implementations, integrate them with the MediaVerse DAM and iterated. According to deliverable D2.2 - Conceptual Design of the MediaVerse Framework, these services are grouped as follows:

- **The Federation Shared Dataspace** provides a distributed storage shared among all the federated network members.
- **The Node Discovery service** allows access to a distributed Node Registry of nodes connected to the federated network.
- **The Federated Search:** Oversees searching and retrieving content within the MediaVerse network. It is responsible for sending search queries and collecting search results across the network, but it delegates the indexing, search, and retrieval of content to the DAM of each node.
- **The Digital Rights Negotiation** Is responsible for facilitating the communication among the different Digital Asset Rights Management (DARM) components of MediaVerse nodes in the federated network, by retrieving the license and copyrights of each content. Moreover, it is responsible for examining the compatibility of different licenses, managing the agreement of multiple collaborators upon a licence, and providing copyright suggestions for users.

The Node Discovery service and the Federated Search services have been merged into a single implementation that within the context of the project is known as IPFS_API.

For the management of Digital Rights Negotiation, the first implementation has been integrated with Digital Asset Rights Management components and a preliminary version of the Federation Shared Dataspace has been enabled to store the asset licenses and copyrights to enable their retrieval from any node (for further details, please refer to section 4.2.4).

The details about the Digital Rights Negotiation functionalities that are not directly related to the communication between nodes will be described in detail in D4.4 - Content Discovery, Copyrights Negotiation Services and Blockchain Repository v2, due by M30 (March 2023).

4.2 Technical Description

This subsection describes the services implemented from a technical point of view, including frameworks used, and granular details about each module.

4.2.1 Basic Infrastructure and Frameworks

The current infrastructure of the service differs from the originally designed one. The main similarity with the first implementation of the service is the programming language used (Python). The rest of the components have been modified as shown in Figure 45.

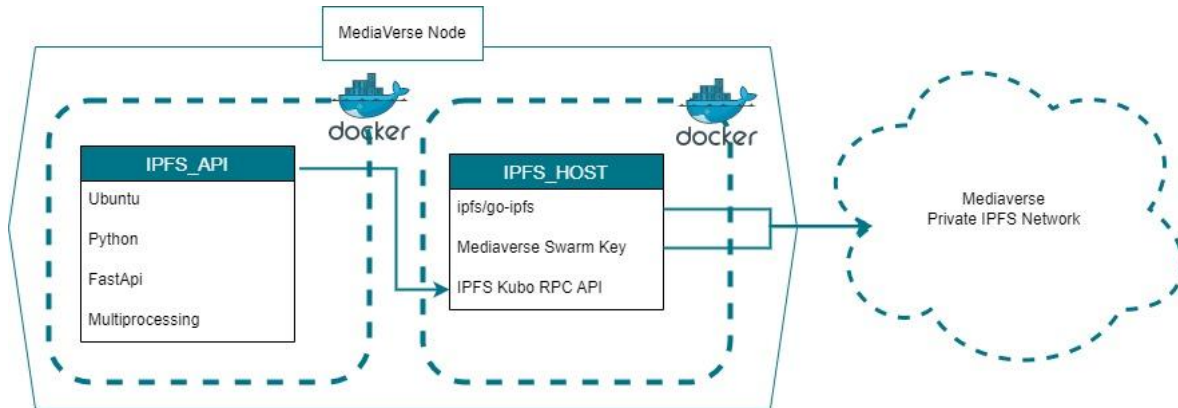


Figure 45: Decentralized Framework Docker infrastructure

Two different containers have been created: IPFS_API and IPFS_HOST. Each MV node joins the MediaVerse Private IPFS network by running an instance of the IPFS_HOST container. The IPFS API provides an abstraction between the rest of the node's micro-services and the Kubo RPC API. This way it takes advantage of the services provided by the IPFS core such as its distributed file system, peer discovery and pubsub experimental implementation. Those two components interact in practically all implementation actions. The communication between them is done using the HTTP protocol, even in the case of interaction with the IPFS pub-sub service, where an HTTP stream is established and remains open. This connection has not been secured, since all the calls between these two components are performed within a private Docker network.

Regarding the frameworks used for each one of the containers:

- **IPFS_API:**
 - **Ubuntu:** The implementation is based on the Ubuntu operating system, which is the most popular platform for containers and the most popular operating system across public clouds and OpenStack clouds¹⁷. We prioritized this container over others also widely used such as Alpine, because although it is a bit heavier, it is a perfectly supportable size for a server like the ones we intend to use in MediaVerse. In addition, the migration of these lighter containers is sometimes tedious as the support and compatibility of the packages they contain is less reliable.
 - **Python:** This choice was made already from the previous implementation. In principle, IPFS has frameworks with native support for JavaScript and Go programming languages, but it was decided to use Python as a reliable and easy to handle language to interact with the Kubo API provided by the IPFS framework.

¹⁷ Docker Hub. (2022). Retrieved 10 August 2022, from https://hub.docker.com/_/ubuntu/

- **Multiprocessing:** This is a Python module that we developed to efficiently manage different separate processes, with control mechanisms, which were not supported by the IPFS Python implementation. The details of these processes will be explained in subsequent sections.
- **Fast API:** This is one of the most popular frameworks for API creation in Python. The support is fully up to date for all the functions required and does not require the installation of a separate module for the creation of the API documentation, which is always updated.
- **Private IPFS Network:**
 - This network is made up of all the nodes that are part of MediaVerse. The communication of the IPFS_HOST instances created throughout the network allows messages to be exchanged privately based on the previously mentioned swarm key. The IPFS_API will send through the local IPFS_HOST a query that will reach the IPFS_HOST instances of all the nodes connected to the network and process their responses. This data flow is explained in more detail in the following sections. This network will also be used to store the licences and copyrights of the asset, so that the information remains available even if the node that created the license disappears.

4.2.2 External Communication

This subsection describes the implementation of the inter-node communication system. This is based on the IPFS implementation of a subscription publishing model, colloquially called pub-sub. This functionality is still under development by IPFS, but after testing, it has been found to be sufficiently mature to perform the tasks we needed for the service.

Pub-sub is an often-used pattern to handle events in large-scale networks. 'Publishers' send messages classified by topic/content and 'subscribers' receive only the messages they are interested in, all without direct connections between publishers and subscribers. This offers much greater network scalability and flexibility¹⁸. Using this subscription publishing model, co-creation of content could be enabled in the future; an external tool could create a topic and subscribe several nodes to it, sharing data transfer in real time.

For the case of inter-node communication, two different types of topics have been defined:

- A global topic called “fsearch” that is shared by all nodes. This will be the receiver of all search requests.
- Another topic that is created by each node with its own ID. This will receive the results of the local search from all the other nodes.

Federated Search

One of the main goals of the Decentralized Framework for MediaVerse communication is to provide a federated search functionality. Figure 46 shows an overview of the current implementation. As an example, the three nodes currently operating in the test network, namely ATC, CERTH and ATOS, are presented.

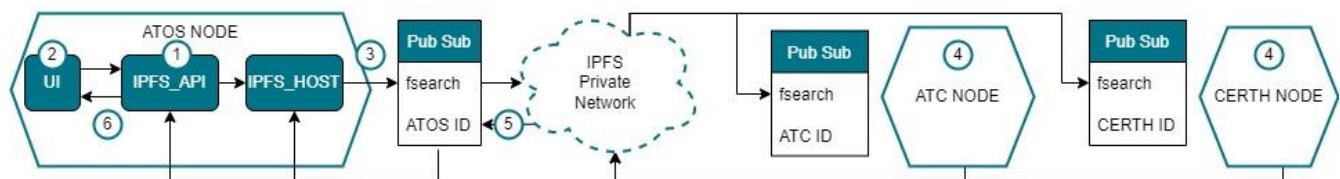


Figure 46: Inter-node pub-sub implementation

¹⁸ For more details on IPFS' pubsub implementation: . Retrieved 11 August 2022, from <https://blog.ipfs.io/25-pubsub/>

1. **IPFS_API initializes the node:** This initialization requires starting a series of processes that are explained in more detail in later sections. In short, the bootstrap node is added to connect to other nodes in the network, the sockets that create the two topics are initialized and a process similar to a garbage-collector for old search results is started.
2. **UI triggers a federated search query:** The interface uses a similar as the one for the local content search, adding the user ID. This call is replicated later by the rest of the nodes. The UI then receives a response example like the following.

```
{
  "id": "a4abf402-1951-11ed-91d6-0242ac150003",
  "timestamp": "1660207396.4132898",
  "user_id": "test_user",
  "term": "test"
}
```

The id provided is later used to look for the results of the user for a federated search, and the timestamp to trigger the results garbage collector to get rid of old searches.

3. **The query is sent to the IPFS network:** This is enabled by the IPFS_HOST and the “fsearch” pubsub topic.
4. **External nodes receive the query:** This query triggers the local DAM of each node, which uses the local Solr index to obtain relevant assets and forwards them to the IPFS_API.
5. **Nodes respond using pub-sub:** Each node asynchronously sends the answer in the topic with the ID of the node that launched the query.
6. **UI periodically calls the IPFS_API to collect results:** The IPFS_API stores the results of each search sorted by user and ID as they come in. The UI periodically calls using the ID of the request to receive the results, and after a timeout, the results for that search are deleted. This asynchronous connection to the results was chosen to avoid scalability problems when a node has a very large number of users.

The result of a federated search of the current integrated implementation is shown in Figure 47.

```
▼ [{nodeID: "12D3KoolWEBaBqodMxAlBPstVUGzErTs8ATdr5YyscvGRJLXFhcs8",...}]
▼ 0: {nodeID: "12D3KoolWEBaBqodMxAlBPstVUGzErTs8ATdr5YyscvGRJLXFhcs8",...}
▼ local_search_result: [{key: "be93acf9-fee7-4318-80c4-d7e6e4dbac7e", score: 1, signature: "dffccb52bc676fd8",...},...]
  ► 0: {key: "be93acf9-fee7-4318-80c4-d7e6e4dbac7e", score: 1, signature: "dffccb52bc676fd8",...}
  ► 1: {key: "e9269563-6305-4132-8c38-6c78e08d6457", score: 1, signature: "dffccb52bc676fd8",...}
  ► 2: {key: "fcbde4de-cd83-4d62-92d4-1f823f544ec8", score: 1, signature: "0d9d31b6172c1e05",...}
  ► 3: {key: "d0bc9a8f-d337-4398-8b65-764fa2a0f385", score: 1, signature: "5362cd996febbb3d",...}
  ► 4: {key: "cfcfe575-3842-4f99-a606-0e68f48d9f58", score: 1, signature: "1e06ff1ee256f50b",...}
  ► 5: {key: "fa645313-37e3-437e-80a3-7a3aba6ddbdb", score: 1, signature: "1e06ff1ee256f50b",...}
  ► 6: {key: "1b8bc3ef-f8c4-49c4-82a7-3a1aeb47ca17", score: 1, signature: "baeb04f0d0fe495d",...}
  ► 7: {key: "a8bd9824-1590-4ae1-94f1-85872f71bc66", score: 1, signature: "9ed7df595c92de52",...}
  ► 8: {key: "21f43dd5-64f9-4fcc-9541-a60a6efc8a0c", score: 1, signature: "b2b25ec37fc2e862",...}
  ► 9: {key: "48f28532-7bd1-4eec-bd5d-0cf9c3a766c2", score: 1, signature: "2566184f8ec52a8e",...}
nodeID: "12D3KoolWEBaBqodMxAlBPstVUGzErTs8ATdr5YyscvGRJLXFhcs8"
```

Figure 47: Grouped Federated search result example

4.2.3 NDD and Future Services Integration

The same mechanism implemented for the federated search is flexible and allows the subsequent integration of other services. For the moment, the near-duplicate detection service developed by CERTH has been integrated in a similar way, and there is a plan to integrate the recommendation service developed by LINKS.

Regarding the NDD service, IPFS is also used to distribute NDD queries across the network in a similar manner as in text queries. NDD supports two types of queries: either the actual media asset for which duplicates are searched or compact feature vectors that describe the asset. For the NDD service, it was decided to transmit the extracted feature vectors instead of the actual content for the following reasons. First, the service is faster as the processing required for feature extraction has already been successfully done when the search is triggered. In addition, as NDD is used for duplicate videos and video segments, sending video queries over IPFS would be inefficient due to the size of that media type. Finally, users are not willing to share their content with other possibly untrustworthy nodes. Sending only the extracted features instead of the content resolves this issue.

The indexing of the content starts from the DAM once an asset is uploaded and published. The NDD service precomputes and stores a set of feature vectors for each asset. Assets are indexed only after a successful license registration, as there is no reason to find duplicates in case of private unlicensed assets. When the federated NDD is triggered for an asset, the query with the extracted feature vectors will be spread over the IPFS network as explained previously for text queries. Each node that listens for incoming NDD queries, requests for duplicates in its local NDD index and duplicate results for each one of the nodes are returned to the node that initiated the process via IPFS. Figure 48 illustrates this process.

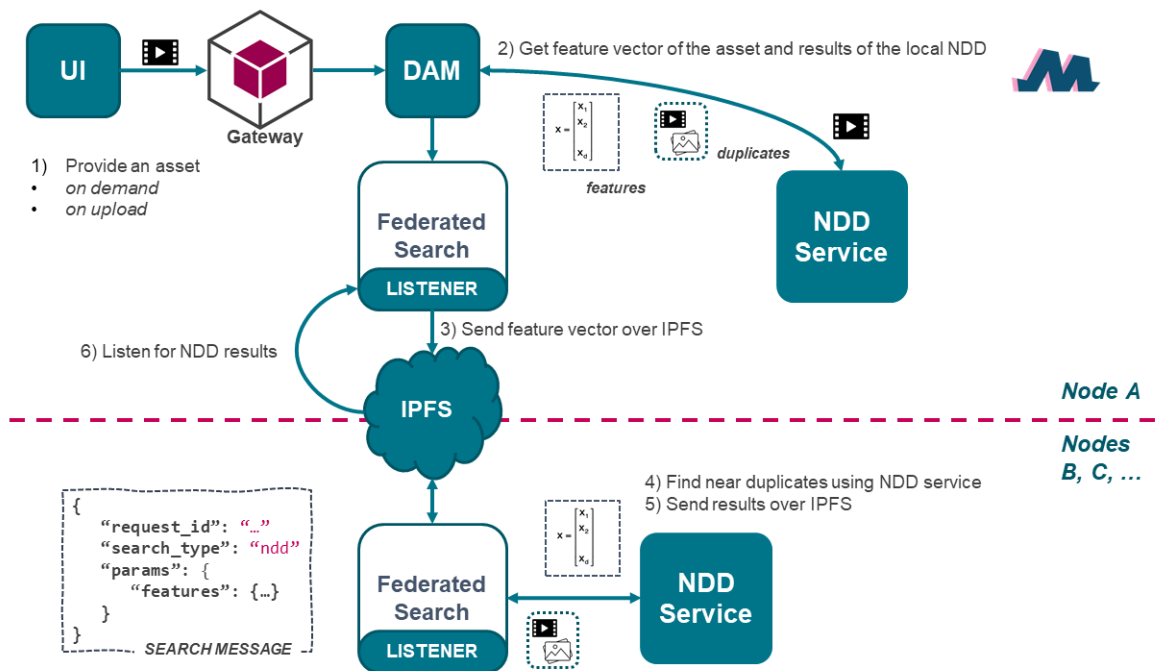


Figure 48: NDD service and Federated Search integration

To integrate other components in the same way, it would only be necessary to add a customized endpoint for the new services, but the transport path is already enabled. The internal structure of NDD, its integration with the rest of the MediaVerse node and the workflows for indexing and retrieval in the context of IPR management is described in detail in deliverable D4.3 - Content Identification services.

4.2.4 SLC Shared Dataspace

To support asset licenses and copyrights retrieval for the Digital Rights Negotiation, it has been decided to store them in the Federation Shared Dataspace provided by IPFS. This decision was made since licenses and copyrights of each asset shall always be available to negotiate the licenses of a Derivative Work (i.e., a work based on already licensed works) regardless of the availability of the node that created the original licenses and copyrights. More details about these procedures will be available in D4.4 - Content discovery, copyrights negotiation services and Blockchain repository v2 (M30). As described in D4.1 - Copyright and Procedures for IPR Definition¹⁹, in MediaVerse the licenses and copyrights of an asset are formalised in a Smart Legal Contract (SLC). Before going into further details, it is important briefly explain the concepts of Ownership Deed and Copyright License:

- **Ownership Deed (OD)** is an SLC where it is stated that the user/s is/are the sole owner/s of the copyrighted work (whether as an author/s or after transfer of such rights).
- **Copyright License (CL)** is an SLC where it is stated that the licensor/s (i.e., the owner/s) license/s to a licensee (i.e., a specific user) the right to use the copyrighted work in a certain way.

Figure 49 gives two examples of these two types of SLC. At an application level, SLCs are JSON objects that contain all the information needed for IPR management, refer to each other and can be stored in IPFS. The IPFS dataspace is perfect for storing SLC objects because it is optimised for files with the same structure that refer to each other. In fact, IPFS uses a Merkle DAG to store content and divides the files into small blocks, storing repetitions in the files only once, and only tracking changes among them.²⁰

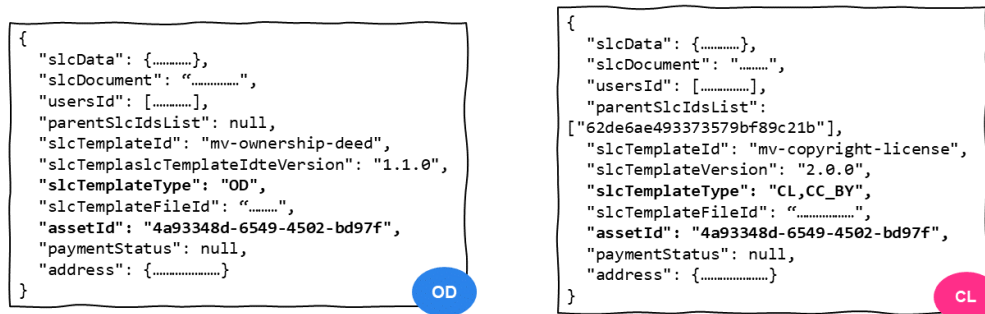


Figure 49 Smart Legal Contracts examples

Figure 50 shows the design and implementation of the storage, retrieval, and management of SLCs on top of IPFS dataspace. This is expected to be integrated in the next versions of the platform. During the license registration for an already stored asset, the DAM first collects parent SLCs stored in IPFS dataspace. The Digital Asset Right Management service receives these requests and generates the appropriate SLC. The DAM that initiated the registration process stores the newly created SLC in IPFS to be accessible by the rest of the nodes in the MediaVerse network. Note that IPFS network does not keep all data available to all nodes instantaneously: only when a request is made to an external node, the requested content is cached. This caching will last until IPFS goes through a periodic garbage collection process. To enable the content to be always available, an extra pinning step is required in the node requesting the content²¹.

¹⁹ https://mediaverse-project.eu/wp-content/uploads/2021/10/MediaVerse_D4.1_Copyright-and-Procedures-for-IPR-Definition.pdf

²⁰ How IPFS works | IPFS Docs. (2022). Retrieved 11 August 2022, from <https://docs.ipfs.tech/concepts/how-ipfs-works/#directed-acyclic-graphs-dags>

²¹ Persistence | IPFS Docs. (2022). Retrieved 11 August 2022, from <https://docs.ipfs.tech/concepts/persistence/#garbage-collection>

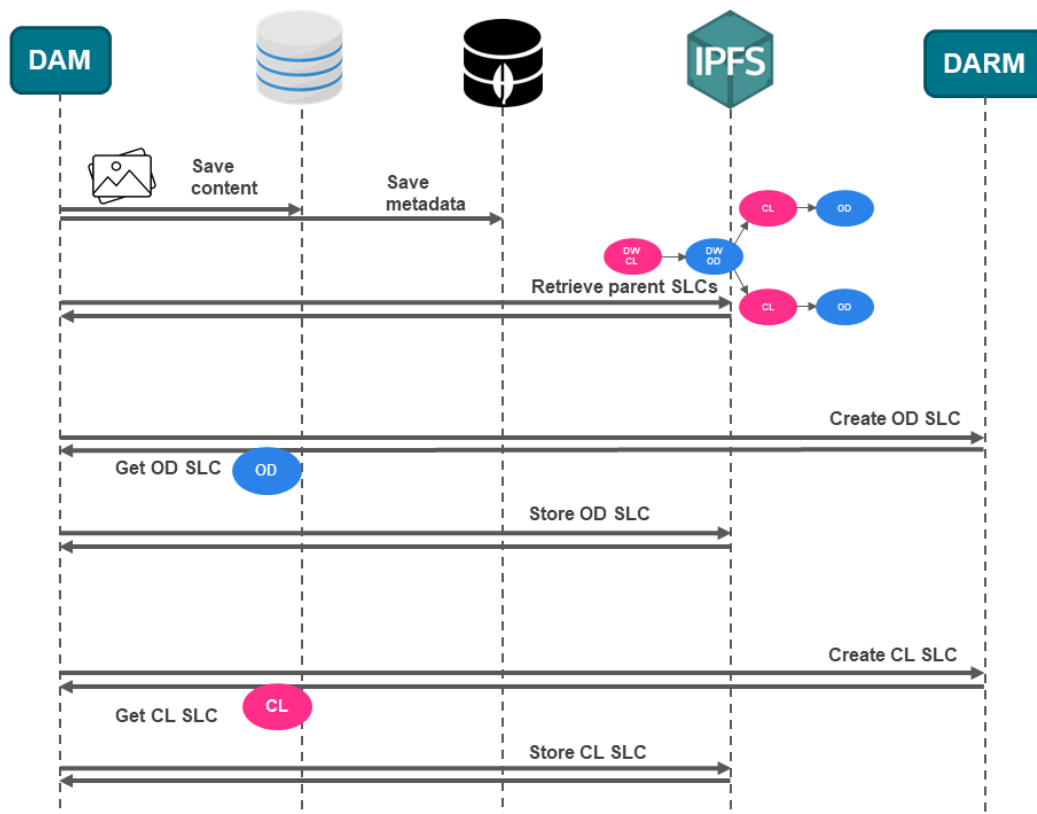


Figure 50 SLC storage in and retrieval from IPFS

For the purchase of assets between nodes, we have designed a solution in which the content and its metadata is copied from one node to another. At the same time the DAM will use IPFS to pin the SLC corresponding to the asset, and in the case of a derivative work, the DAM will pin also all the SLCs of its parent assets. This ensures the availability of the SLC of all the imported assets in a node.

- After the OD creation, the SLC provided by the IPR service is a JSON object with information about the licenses, the user, the smart contract ID, etc. (it will not be shown here due to its size).
- Once the DAM receives the SLC, it will store the JSON file and add it to the IPFS through the IPFS_HOST, which will automatically pin the JSON in the local node, enabling one of the call parameters. This will return a hash referencing the file:

```
{
  "Name": "slc.json",
  "Hash": "QmaUrqYoahnKCyAsRHLrcz5PZJtBRGWFT7kgaBYvQurFDk",
  "Size": "19"
}
```

This hash will be referenced in the DAM asset model.

- When a different node DAM instance needs to pin the information of a SLC at the time of acquiring an external asset, it will call its local IPFS_HOST instance to pin the hash provided in the desired asset data model. The SLC information is stored persistently in both nodes, ensuring the persistence in the network. If an asset is popular and IPFS will take advantage of the files fragmentation to download the SLC from several nearest nodes simultaneously using the underlying p2p technology, resulting in fast retrieval.
- The SLC file content of any file in the MediaVerse Network can be retrieved through the local IPFS_HOST using the hash reference in the asset model.

4.2.5 Internal Architecture

The previous sections focused on the interactions of the IPFS_API with the external microservices of the platform, this section focuses on the functions and processes developed within the IPFS_API (Figure 51).

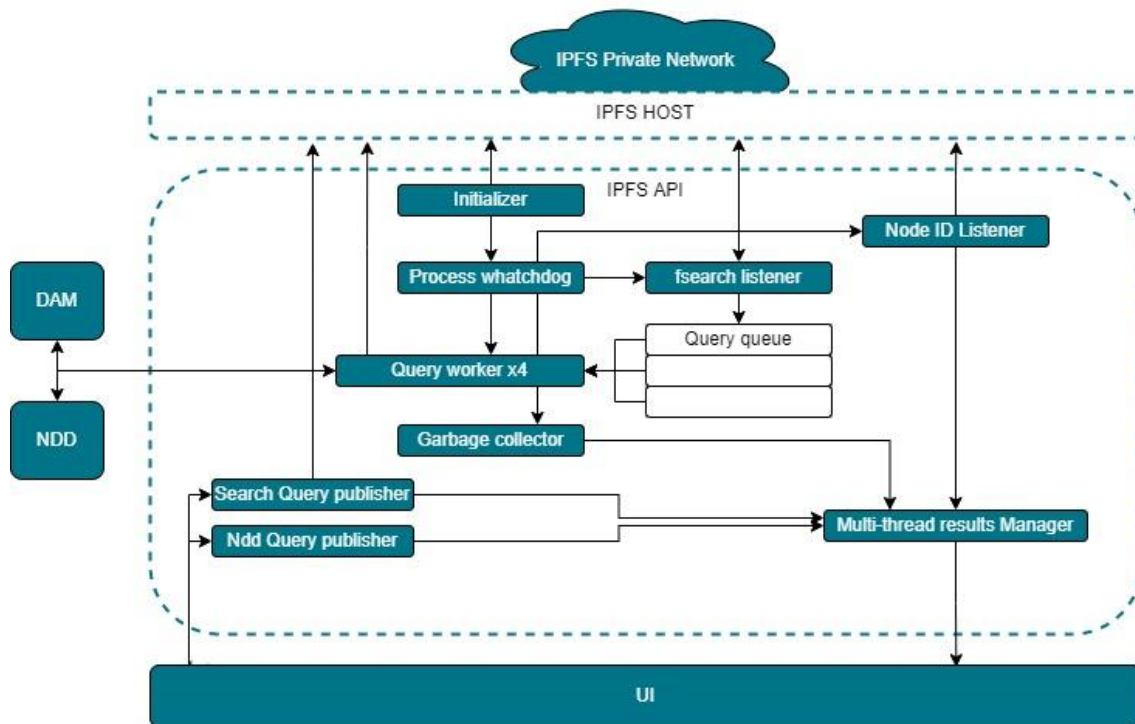


Figure 51: IPFS_API Internal architecture

The following processes make up the IPFS_API:

- **Initializer:**
 - Sets up the environment variable provided in the .env file.
 - Waits for the availability of the IPFS_HOST container.
 - Once available, removes the default bootstrap list provided at IPFS initialization and adds the bootstrap address included in the environment variable. The bootstrap node will later retrieve the addresses of the peers in the network. In the first proof of concept this bootstrapping process was not automatic and had to be done after deployment. This is why this initializer was added.
 - Initializes the process watchdog.
- **Process watchdog:** To make the solution more robust this process was included to monitor the rest of the independent processes and launched them again in case any of them is not available.
- **fsearch listener, Node ID listener:** These two processes open a socket connection against the IPFS_HOST to create their corresponding topics and keep listening for new messages. This solution makes these the only open sockets between two micro-services within a MediaVerse node. When a query is received, the fsearch listener stores it in a Query queue to be consumed by the workers.
- **Query worker:** These processes trigger the received queries against the corresponding services. As the processes that handle the queries are sometimes heavy, response times can be long. By dividing this task among several worker processes, the service will be able to support more simultaneous calls. After a Query is processed, the result is published in the IPFS topic of the requesting node.

- **Multi-Thread Results Manager:** The results of the federated search must be accessed as fast as possible. Furthermore, using several independent processes to write to the same place can lead to race conditions. For these reasons, it was decided to use a Multiprocessing Manager to create a structure that would be stored in volatile memory and avoid race conditions.
- **Garbage Collector:** Service responses can also be heavy and will accumulate over time, so if the Results Manager volume was not controlled, they could take up a large amount of memory space. Therefore, this garbage collector is regularly run to delete the searches that have been previously performed.

4.3 Functional Description

This subsection describes the interface the IPFS_API. Figure 52 presents the corresponding Swagger interface.

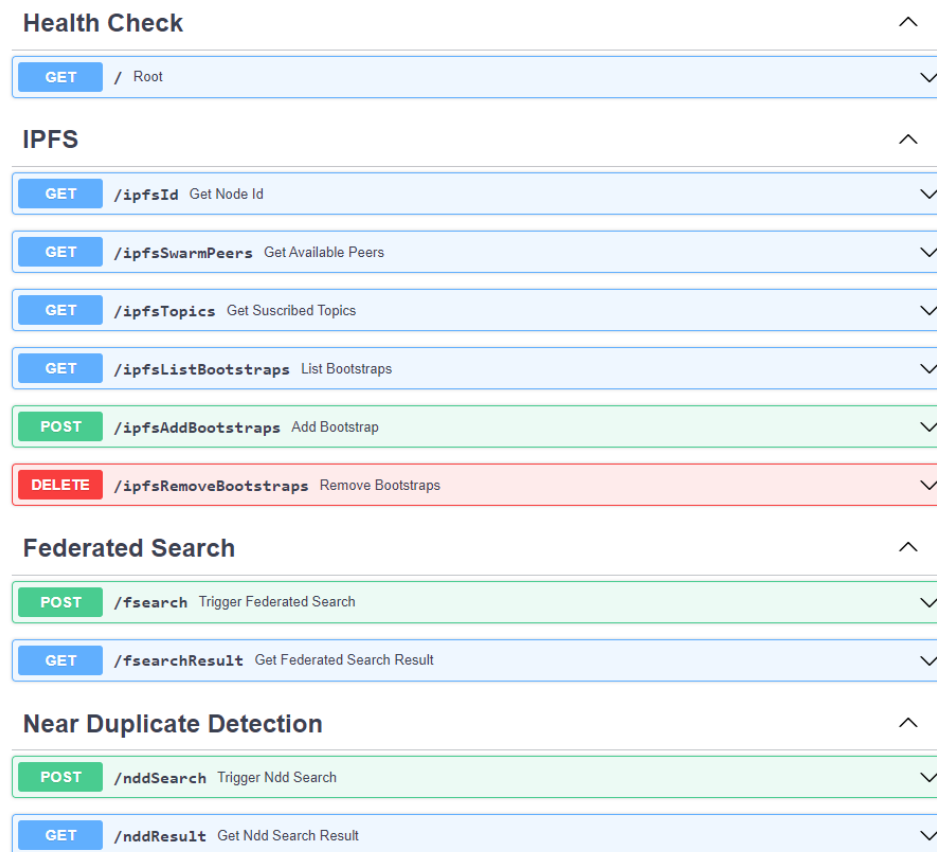


Figure 52: IPFS_API documentation - swagger interface

The interface is tagged according to its functionality:

- **Health Check:** This endpoint is intended to serve a future external monitoring service.
- **IPFS:** This groups endpoints that are useful for IPFS troubleshooting purposes. The user can consult the information of their node in relation to IPFS. It lists the topics to which the node is subscribed, which should be the two mentioned in previous sections. Finally, it interacts with the IPFS bootstraps to add or delete new ones without having to restart the service.
- **Federated Search:** It contains the calls to perform the federated search and obtain the results using the ID provided in the first call. As shown in Figure 53, search filters are in accordance with the DAM.
- **Near Duplicate Detection:** It contains calls to the NDD service, one to perform a federated NDD, and another to receive the results.

Federated Search

POST /fsearch Trigger Federated Search

Triggers the federated Search

Parameters

Name	Description
user_id * required string (query)	<input type="text" value="user_id"/>
term * required string (query)	<input type="text" value="term"/>
sort array[string] (query)	<div>Add string item</div>
page integer (query)	<input type="text" value="page"/>
per_page integer (query)	<input type="text" value="per_page"/>
media_type string (query)	<input type="text" value="media_type"/>
since integer (query)	<input type="text" value="since"/>
until integer (query)	<input type="text" value="until"/>
is_meme boolean (query)	<div>--</div>
is_disturbing boolean (query)	<div>--</div>
objects array[string] (query)	<div>Add string item</div>
min_faces integer (query)	<input type="text" value="min_faces"/>

Cancel

Figure 53: IPFS_API interactive documentation detail

The final result of the integration of the service in the MediaVerse node would be the representation of the assets of other nodes in the web interface. Figure 54 shows the current version of the interface.

Local Search

External Search

Node: mediaverse.ari-imet.eu

FILE

VolkswagenGTIRewiew_cut-sub.s.mp4

Type: video

Author: airan

a month ago

VolkswagenGTIRewiew_cut-sub.s.mp4

Type: video

Author: airan

a month ago

Cravendale, Cats with Thumbs.mp4

Type: video

Author: test0020@mail.com

a month ago

Figure 54: Federated search results in the UI

4.4 Deployment

In the latest version, all configuration files are included inside the “.env” file and are triggered from the IPFS_API at start-up. This way, the deployment can be done by running a single docker-compose command. Before running the docker-compose up, the environment variables must be set:

- **config/ipfs_host/.env**

```
LIBP2P_FORCE_PNET=1
IPFS_PROFILE=server
IPFS_SWARM_KEY_FILE=/data/ipfs/myswarm.key
```

When running the solution from the source code these variables are part of the IPFS configuration and should be left untouched. In case of the deployment from the prebuilt image, this “.env” file will not be necessary as the configuration will be already present in the IPFS_HOST container.

- **LIBP2P_FORCE_PNET** ensures the creation of the private network.
- **IPFS_PROFILE** sets the host to discover external peers.
- **IPFS_SWARM_KEY_FILE** sets the location of the swarm key inside the container.

- **config/ipfs_api/.env**

```
IPFS_NODE_IP=ipfs_host
IPFS_NODE_PORT=5001
IPFS_BOOTSTRAP_ADDR=
IPFS_NODE_TIMEOUT=10
FSEARCH_RESULT_TIMEOUT=30
DAM_ADDR=
NDD_ADDR=
MAX_WORKERS_NUM=4
EXPOSE_CONTENT=True
```

These variables should be set based on the configuration requirements of the node administrator:

- **IPFS_NODE_IP**, **IPFS_NODE_PORT**, **DAM_ADDR**, **NDD_ADDR** These variables store the endpoints the IPFS_API needs to contact local node services. In case of MediaVerse deployment, they will correspond to service names and ports declared inside the Docker compose configuration.
- **IPFS_BOOTSTRAP_ADDR** must have the format: `/ip4/<External IP addr>/tcp/4001/p2p/<Node ID>` and should be taken from the Bootstrap node ‘s response to the `/ipfsId` call.
- **IPFS_NODE_TIMEOUT** is the time IPFS_API waits for IPFS_HOST to be ready.
- **FSEARCH_RESULT_TIMEOUT** is the time Query workers wait for responses.
- **MAX_WORKERS_NUM** is the number of query workers. This could be increased to respond to multiple requests simultaneously but could lead to increased CPU consumption.
- **EXPOSE_CONTENT** if set to False, this node will not share its content in the federated searches of the rest of the nodes.

The path to the environment variables should be the same whether starting the containers from the general MediaVerse docker-compose or creating them from the source code.

A docker-compose up command will initialize both containers. If successfully initialized, the last logs should report the processes created.

```
mv_ipfs_api | 2022-08-12 08:45:25 DEBUG fsearch_functions id_listener Initializing node ID subscription. PID=16
mv_ipfs_api | 2022-08-12 08:45:25 DEBUG fsearch_functions fsearch_listener p Initializing fsearch subscription. PID=19
mv_ipfs_api | 2022-08-12 08:45:25 DEBUG fsearch_functions reader_from_fsearch fsearch_reader worker started PID= 20
mv_ipfs_api | 2022-08-12 08:45:25 DEBUG fsearch_functions reader_from_fsearch fsearch_reader worker started PID= 25
mv_ipfs_api | 2022-08-12 08:45:25 DEBUG fsearch_functions reader_from_fsearch fsearch_reader worker started PID= 27
mv_ipfs_api | 2022-08-12 08:45:25 DEBUG fsearch_functions reader_from_fsearch fsearch_reader worker started PID= 29
```

Figure 55: IPFS AP successful initialization logs

5 Component Development, Integration and Deployment

A MediaVerse node consists of a set of Docker containers (Figure 56). These contain core functionalities of each node, such as user authentication and media asset management, as well as services integrated in the MV node architecture, such as federated search and retrieval, IPR management, and media asset publishing.

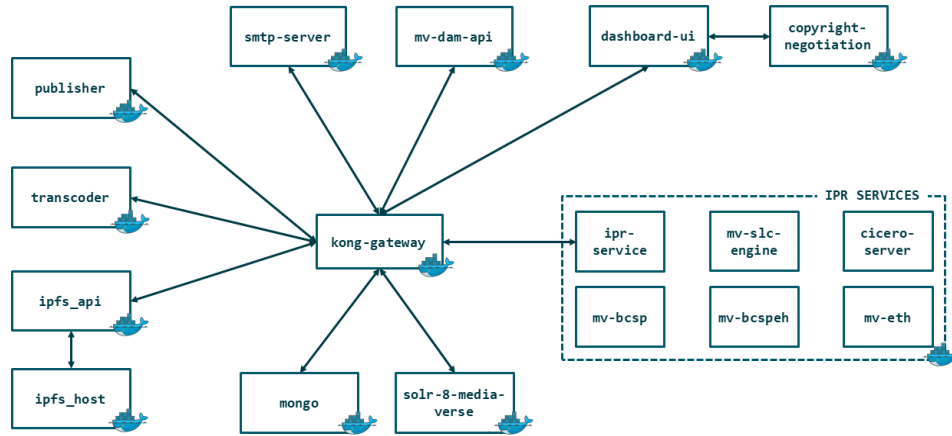


Figure 56. Docker Containers

5.1 Feature Development Process

For development, two Gitlab boards of issues are used: features and development. The first board contains all the functional requirements that are available and those that must be implemented (Figure 57).

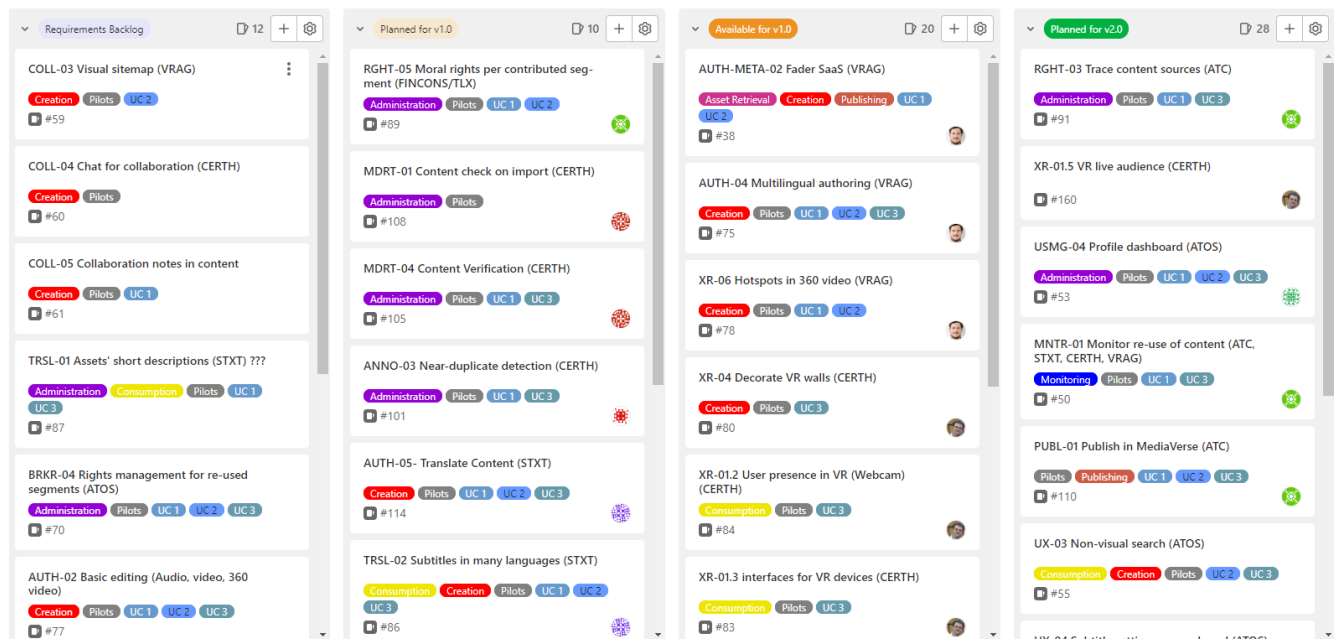


Figure 57: The Features board

The second board is the translation of the functional requirements to technical tasks that must be implemented. The board consists of six columns: Backlog, Suggestions, Design, Bugs, In Progress and Closed (Figure 58 and Figure 59).

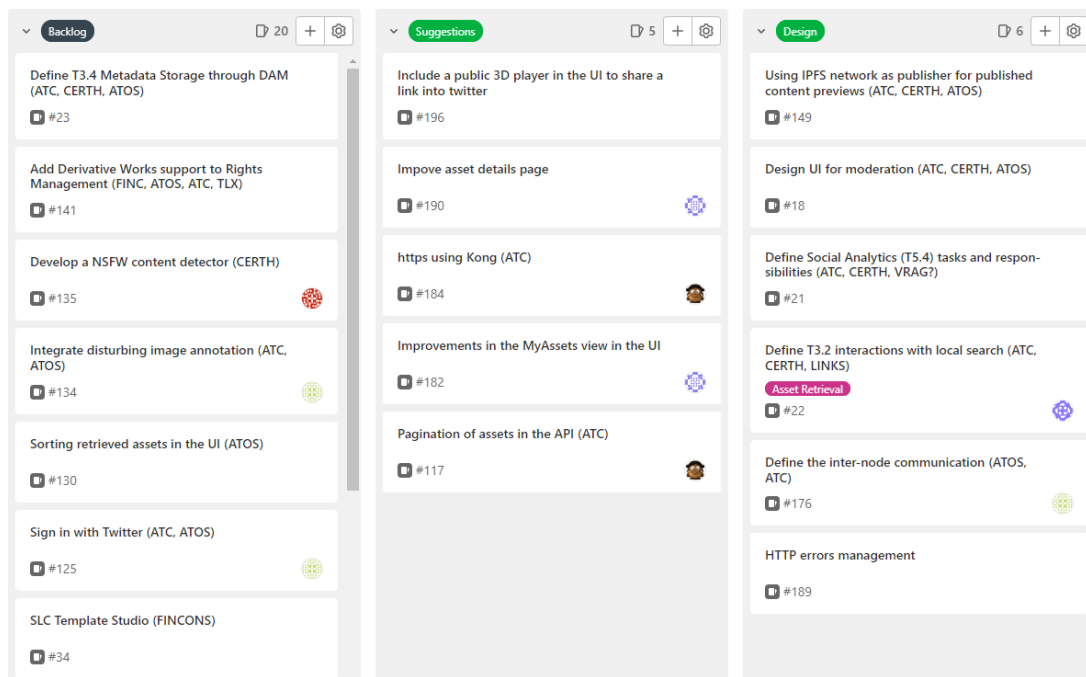


Figure 58: Board of Development issues

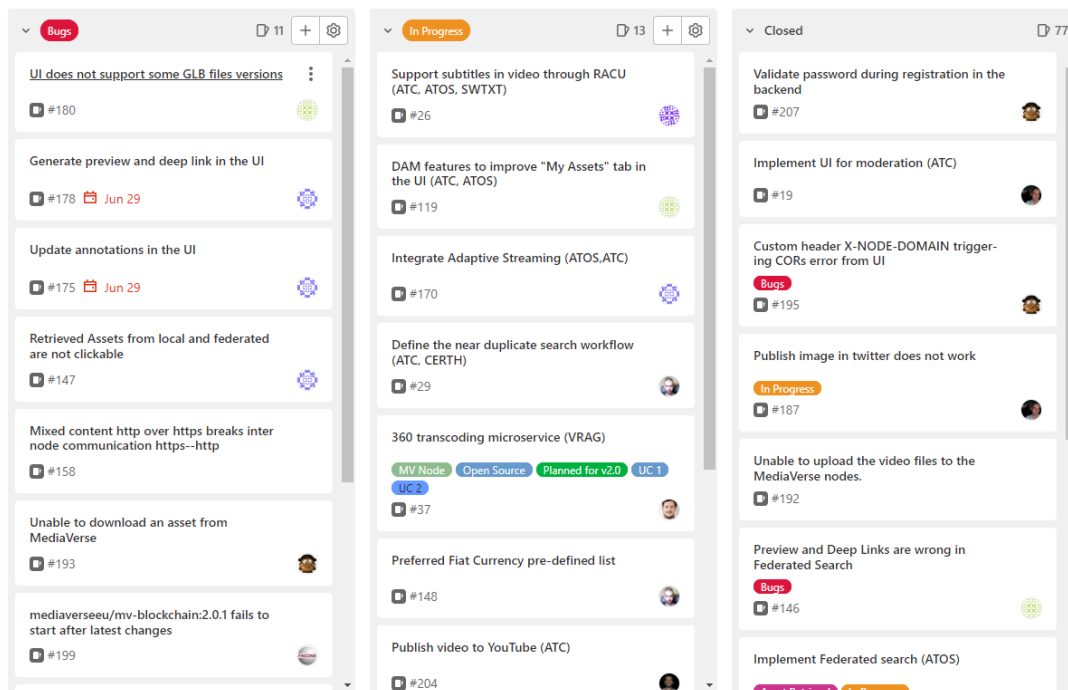


Figure 59: Board of Development issues

The development process starts with the Design phase where technical issues derived from the functional requirements are added in that category. After the Design phase, the task is moved in the Backlog (low priority or future work) or In the Progress column where the actual development takes place. Finally, when the technical task is implemented and deployed, it is moved in the Closed category. There are also two columns, Suggestions and Bugs, where any partner can either propose “nice to have” features or report a bug respectively.

5.2 Integration

Software management in MediaVerse is using semantic versioning for each component that is integrated in the MV node. Every version of each module is stored in a dedicated docker hub repository²² as a docker image (Figure 60). The latest versions of the above images are then included in the docker compose configuration file that contains the instructions for creating all the available resources for the MediaVerse node. Whenever a new version of a component is available, it is dockerized, uploaded to the docker hub and a new merge request in the respective Gitlab repo²³ is triggered for the update of the docker compose file. In that way, the docker compose configuration is reviewed and it is available for cloning in the MediaVerse nodes.

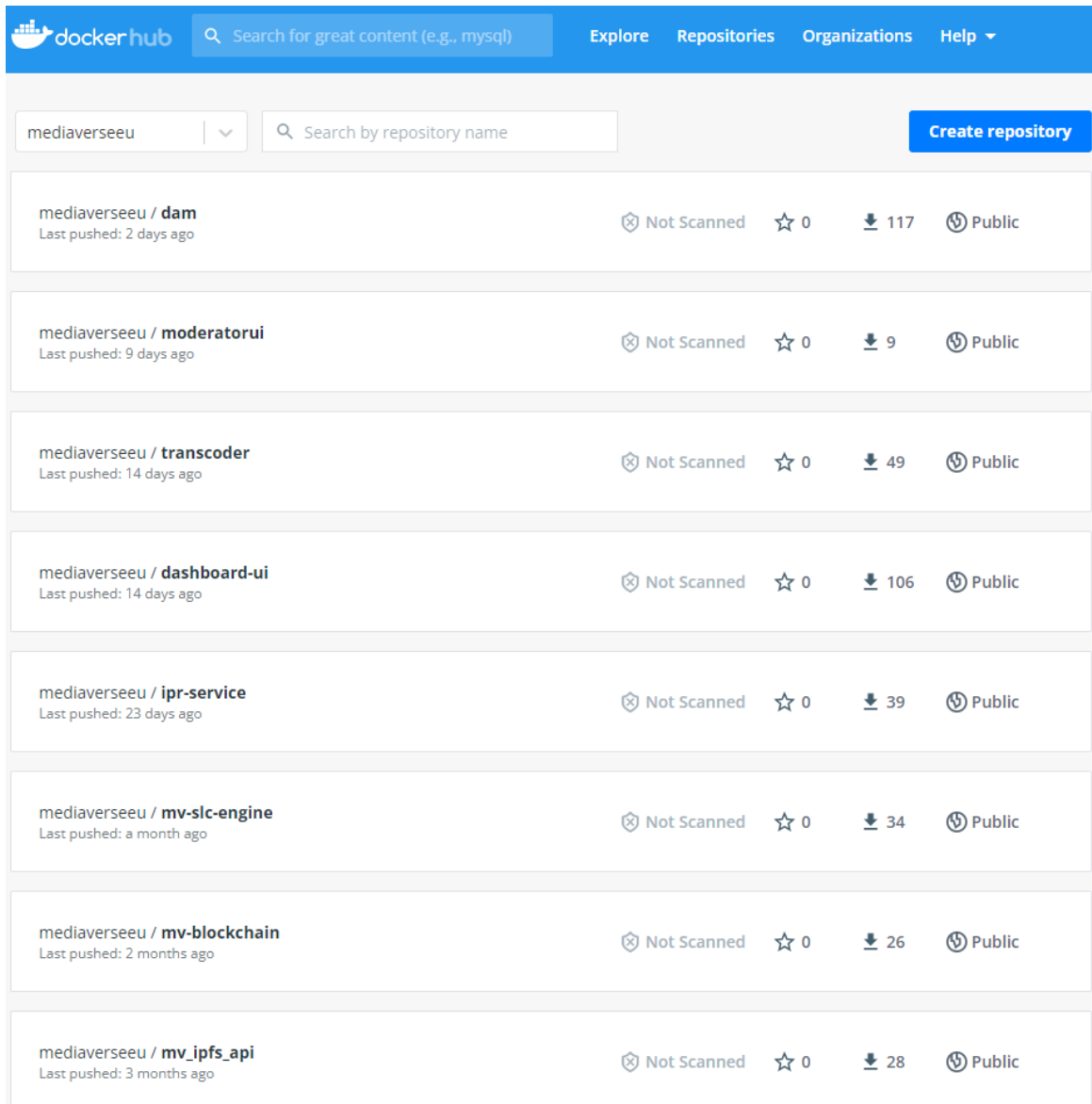


Figure 60: A view of docker hub repository

²² <https://hub.docker.com/u/mediaverseeu>

²³ <https://gitlab.com/mediaverse/wp6-group/wp6-docker-compose-configuration/-/blob/main/Deployme/config/solr/cores/mediaverse/managed-schema>

5.3 Deployment

The deployment structure consists of a **docker-compose** file, a **mongo init script** and a **config folder**. The docker-compose file, the init-mongo.js and the config folder must be placed in the root level.

1. The docker-compose file contains all the docker images, networks and volumes needed for the docker engine to deploy the MV node. It is predefined and it is common for all nodes, but node administrators can edit it accordingly to change the port in which each service is exposed.
2. The mongo init script will be called by the docker-compose file to properly initialize the database. It is predefined and it is common for all the nodes.
3. The config folder has the following structure.

config
..dam
..ipfs_api
..ipfs_host
..kong
..moderationui
..mongo
..postfix
..right-management
..solr
..ui

This folder includes all the necessary configuration files of the node. Installation instructions and configuration details for the deployment of a MV node can be found in the GitHub MediaVerse repo²⁴.

²⁴ <https://github.com/MediaVerse-Project/mediaverse-node>

6 Conclusion and Next Steps

This deliverable described in detail the updates concerning the MediaVerse implementation, with a particular focus on the DAM, the Dashboard UI and the Federated Search. It also provided information on the development and deployment approach that we have adopted in the project.

During the third year, the following steps are planned:

- Addition of new functionalities to the UI:
 - Improved integration of Near Duplicate Detection (cf. section 4.2.3)
 - Integration of other media repositories (e.g. Wikimedia)
 - Addition and optimization of 360 videos
 - Addition of SMTP email for MV users
- UX improvement, including usability and accessibility improvements based on partner's feedback.
- Continuous integration of new services, external and internal.
- Refinement of integration between the MV private IPFS network and license management services.
- Integration of other developed services that require federated search.
- Testing with actual media content once the NDD service is fully integrated.
- Execution of performance tests in a three-node setup, to evaluate the true high availability of using IPFS by simulating a high load scenario.



MediaVerse is an H2020 Innovation Project co-financed by the EC under Grant Agreement ID: 957252.
The content of this document is © the author(s). For further information, visit mediaverse-project.eu.