Universidade Federal
do Rio de Janeiro
Escola Politécnica

uMAM, UMA PROPOSTA DE MAM ABERTO

Yuri Vasquez Fernandes

Projeto de Graduação apresentado ao Curso de Engenharia de Computação e Informação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientadores: Cláudia Maria Lima Werner
Sérgio Nazaré de Sá Duque
Estrada Meyer

Rio de Janeiro
Julho de 2019

uMAM, UMA PROPOSTA DE MAM ABERTO

Yuri Vasquez Fernandes

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE COMPUTAÇÃO E INFORMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE COMPUTAÇÃO.

Examinado por:

Prof. Cláudia Maria Lima Werner, Ph.D.

Prof. Sérgio Nazaré de Sá Duque Estrada Meyer, M.Sc.

Prof. Claudia Susie Camargo Rodrigues, D.Sc.

Prof. Edilberto Strauss, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
JULHO DE 2019

Vasquez Fernandes, Yuri

uMAM, uma proposta de MAM aberto/Yuri Vasquez Fernandes. – Rio de Janeiro: UFRJ/ Escola Politécnica, 2019.

XII, 42 p.: il.; 29, 7cm.

Orientadores: Cláudia Maria Lima Werner
Sérgio Nazaré de Sá Duque Estrada Meyer

Projeto de Graduação – UFRJ/ Escola Politécnica/ Curso de Engenharia de Computação e Informação, 2019.

Referências Bibliográficas: p. 40 – 42.

1. MAM. 2. TV. 3. Produção de Conteúdo. I. Lima Werner, Cláudia Maria *et al.* II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia de Computação e Informação. III. Título.

Dedico este trabalho aqueles que emprestam seus olhos e ouvidos para que possamos ver e ouvir o mundo.

Agradecimentos

Agradeço a minha mãe, Léa, por tudo que me ensinou e que serviu de base para a vida acadêmica e todo o resto. Agradeço, também, a minha família pelo apoio ao longo destes anos.

Agradeço a minha esposa, Juliana, pelo carinho, pela paciência, por me inspirar nos momentos difíceis e por proporcionar tantos momentos felizes.

Agradeço aos meus amigos que sempre estiveram por perto e que também me ajudaram a superar as dificuldades da vida acadêmica. Agradeço especialmente ao Rafael Romeiro, Gustavo Pfeiffer e Gabriel Mendonça que sempre tiveram tanta paciência em me ajudar sem pedir nada de volta.

Agradeço ao Igor Rocha, Lucas Laurentino, Lili Anjos e ao meu amigo Rodrigo Amorim por ajudar a reaver qualquer chance de eu me formar. Sem vocês não seria possível a minha graduação, obrigado!

Agradeço a todos os professores que me ensinaram e me inspiraram ao longo da vida acadêmica.

Agradeço aos meus orientadores Claudia e Sérgio por abraçar o projeto, pela paciência e pela orientação cuidadosa e atenciosa.

Agradeço também a todos os meus colegas de trabalho que contribuíram para a minha evolução sem a qual não seria possível desenvolver o projeto.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/ UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação.

uMAM, UMA PROPOSTA DE MAM ABERTO

Yuri Vasquez Fernandes

Julho/2019

Orientadores: Cláudia Maria Lima Werner
Sérgio Nazaré de Sá Duque Estrada Meyer

Curso: Engenharia de Computação e Informação

A utilização de sistemas de MAM (*Media Asset Management*) é fundamental para os fluxos de produção de conteúdo audiovisual na TV. Com o surgimento de novas plataformas de distribuição de vídeo, surgem, também, novos produtores de conteúdo, trazendo consigo, novos processos de produção. Neste trabalho, são levantadas as características comuns aos MAM's existentes e é proposto o desenvolvimento de um MAM voltado para estes novos produtores de conteúdo.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Engineer.

uMAM, AN OPEN MAM PROPOSAL

Yuri Vasquez Fernandes

July/2019

Advisors: Cláudia Maria Lima Werner
Sérgio Nazaré de Sá Duque Estrada Meyer

Course: Computer Engineering

The use of MAM (Media Asset Management) systems is fundamental for content production workflow on TV. With the emergence of new video distribution platforms, new content producers also emerge bringing with them new production processes. In this work common characteristics of the existing MAM's are collected and a new MAM aimed at those new producers is developed.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Neste capítulo são apresentados os conceitos abordados e a estrutura do trabalho. Para tal foi dividido em 3 seções. A Seção 1.1 introduz o tema do MAM (*Media Asset Management*). A Seção 1.2 apresenta os novos desafios na produção e distribuição de conteúdo multimídia que motivam o desenvolvimento do sistema proposto no trabalho. Por fim, a Seção 1.3 apresenta a estrutura do trabalho.

1.1 Tema

O surgimento, na década de 1920, da tecnologia de TV por radiodifusão, e sua popularização, após o fim da Segunda Guerra Mundial [5], provocaram uma reconfiguração na comunicação de massa, antes dominada pelo rádio e mídia impressa [6]. Com a TV era possível transmitir conteúdo multimídia para milhares de pessoas de forma síncrona, ao mesmo tempo, trazendo inúmeras possibilidades para o jornalismo, publicidade, dramaturgia e entretenimento.

A chegada da tecnologia do *videotape* em 1957 no mercado americano, marcou o declínio do uso de cinescópios para armazenamento e reprodução de conteúdos gravados e editados, já que permitia o armazenamento de conteúdos, magneticamente, em maior qualidade que seu antecessor [5]. Com isso, a utilização de conteúdo gravado e pós-produzido recebeu maior adoção nas programações televisivas.

Com a necessidade de gerenciar as mídias (outrora físicas) utilizadas nos processos de produção de vídeo (principalmente na TV) e com o aumento na capacidade computacional de processamento e armazenamento de dados, a produção de conteúdo para a TV se viu numa passagem de paradigma do uso de tecnologia analógica para o digital. Nesse contexto, os sistemas de catalogação de fitas (semelhantes àqueles usados em bibliotecas) seriam substituídos por sistemas de gerenciamento de ativos (*assets*) de mídia (*Media Asset Management*, ou MAM) que passariam a ser baseados em arquivos digitais [7].

Um sistema de MAM para os fluxos de produção de conteúdo (jornalístico, esportivo ou de entretenimento) tem como premissa garantir a colaboração e agilidade no processo de produção de conteúdo, assim como, qualidade e conformidade do conteúdo disponibilizado com o padrão de sua organização. Além disso, ele deve ser capaz de gerir o ciclo de vida de seus ativos, a fim de melhor aproveitar o espaço de armazenamento disponível, diminuindo, dessa forma, os custos de produção em larga escala [7].

Com a chegada da TV digital, os MAMs passam a ter um papel fundamental na

garantia da qualidade do conteúdo na transição do SD (*standard definition*), cuja relação de aspecto é de 4:3 e cuja resolução varia entre 480i (padrão Americano) e 576i (padrão Europeu), para o HD (*high definition*), cuja relação de aspecto é de 16:9 e cuja resolução varia de 720p a 1440p [7].

1.2 Motivação

O surgimento, em 2005, do Youtube e a consolidação das plataformas de distribuição de conteúdo em vídeo pela internet como Twitch[1], Facebook, Snapchat[2], e a introdução de plataformas de *streaming* como Netflix, Amazon Prime, HBO Go, Globoplay, etc, os modelos de exibição, transmissão e produção de conteúdo passam a sofrer profundas mudanças. Mais especificamente, em plataformas que permitem um leque mais amplo de publicadores, a produção de conteúdo tende a se tornar menos centralizada, uma vez que o alcance do conteúdo beneficia a diversidade de produtores nestas plataformas, e.g., uma produtora de conteúdo de algum nicho que não possui uma emissora de TV não conseguiria alcançar o seu nicho. Além disso, essas plataformas transmitem conteúdo pela internet, permitindo investimentos estruturais menores (já que não são elas que mantêm a internet) e distribuem em diversos dispositivos, criando novos padrões de consumo de conteúdo.

Além disso existem plataformas digitais independentes, como é o caso da COPPETV [8] cujos produtores de conteúdo são independentes e possuem um nicho específico.

Isso não significa, porém, que as necessidades de colaboração, qualidade e velocidade tenham diminuído. A diferença é que agora a produção de conteúdo também precisa ser mais distribuída. Diante de tais fatos, enxergou-se a necessidade da criação de uma plataforma aberta e acessível que permita a produção de conteúdos de forma colaborativa, de qualidade, com velocidade e que atenda pequenas organizações ou indivíduos que produzem conteúdo audiovisual.

Neste trabalho é proposta uma plataforma desta natureza, abordando as especificações das funcionalidades elicitadas (no Capítulo 2), assim como, o processo de modelagem e implementação.

1.3 Estrutura do Trabalho

Este trabalho está organizado da seguinte maneira:

- Capítulo 1 - Introdução - Revela a relevância do tema abordado e expõe a motivação do trabalho, além da estrutura do trabalho.
- Capítulo 2 - Estudo do Modelo Atual - Levanta as características das ferramentas existentes.
- Capítulo 3 - Tecnologias utilizadas - Descreve tecnologias, cuja utilização foi relevante no desenvolvimento do trabalho.
- Capítulo 4 - Solução proposta - Define os objetivos e funcionalidades da ferramenta desenvolvida.

[1] https://www.twitch.tv/ - Plataforma de compartilhamento de vídeos ao vivo de jogos
[2] https://www.snapchat.com/ - Plataforma de compartilhamento de mídias curtas e fugazes

- Capítulo 5 - Desenho da solução - Descreve os detalhes de alto nível do projeto da ferramenta.
- Capítulo 6 - Decisões de implementação - Esmiúça os detalhes de implementação das bibliotecas/ferramentas desenvolvidas.
- Capítulo 7 - Modelos de operação - Descreve alguns modelos de operação possíveis para a implantação e manutenção do sistema.
- Capítulo 8 - Conclusão - Traz conclusões sobre o trabalho desenvolvido e algumas propostas de melhorias e evoluções para a ferramenta.

Capítulo 2

Estudo do Modelo Atual

Para definir as características necessárias para o desenvolvimento de um MAM, é preciso estudar os padrões e funcionalidades já aplicados nos sistemas existentes. A fim de entender melhor o modelo atual de MAM praticado pela indústria de TV, foi realizada uma pesquisa dos principais players do mercado, Avid e VizRT. Devido à ausência de fontes acadêmicas ou até mesmo de materiais técnicos disponíveis, foi necessário realizar a pesquisa pela leitura dos websites dos sistemas e observação dos mesmos.

Com o objetivo de fazer uma comparação mais clara entre eles, são descritas as principais funcionalidades comuns na Seção 2.1 e, então, suas características individuais são ressaltadas na Seção 2.2. Na Seção 2.3, é apresentada uma visão do que é oferecido de código aberto em termos de MAM. A Seção 2.4 apresenta uma breve introdução ao MXF, formato de contêiner de vídeo presente em todos os fluxos de produção de TV. Finalmente, a Seção 2.5 traz algumas conclusões sobre a pesquisa realizada.

2.1 Principais Funcionalidades

Durante a pesquisa sobre modelos de mercado, foram encontradas algumas funcionalidades comuns entre os MAM, descritas a seguir.

2.1.1 Busca de mídias

Como o principal propósito de um MAM é permitir acesso a um repositório de mídias para produção de conteúdos, é imprescindível a existência de um ou mais mecanismos de busca para possibilitar a localização e obtenção de materiais. Sendo assim, os MAMs têm a busca como peça central para agilizar o processo de produção. É possível, por exemplo, buscar conteúdos procurando por etiquetas (*tags*) que os identifiquem, buscando pela palavra "crime", por exemplo.

2.1.2 *Logging*

Ao processo de marcação de pontos (no tempo) com um ou mais rótulos, dá-se o nome de *logging* [7]. Este processo, seja automático ou manual, é muito importante para a recuperação de mídias relevantes para uma determinada produção, e.g., mar-

car um gol numa partida de futebol. Está presente em MAMs, seja internamente, ou através de ferramentas externas.

2.1.3 Processamento e normalização de entradas de mídia

Para garantir a manutenção e melhoria da qualidade do conteúdo de seu acervo, o MAM deve possibilitar a entrada de materiais de diversas fontes. Para que isto ocorra sem que haja prejuízo para a qualidade do material no sistema (mídias), o MAM conta com workflows responsáveis pela normalização do material de entrada, seja pela transcodificação, seja pelo reempacotamento das mídias. O MAM pode receber, por exemplo, um arquivo, de uma câmera de telefone celular, com resolução, taxa de quadros e taxa de bits diferente do padrão da organização, que tornaria o uso daquele material inviável, e normalizá-lo para o formato padrão. Esta normalização na entrada permite que o processo de edição de vídeo ocorra sem conversões manuais que parariam o processo.

Além disso, para que a busca e pré-visualização não sejam tarefas custosas para o sistema, são gerados um formato de alta qualidade, para a entrega final, e um formato de baixa qualidade para a pré-visualização. Caso o padrão de alta resolução possua uma taxa de bits de 50 Mbps e o formato de baixa, 8 Mbps, há uma economia de 84% nos custos de transferência para fora (*Data transfer out*) em ambientes de nuvem, como é o caso da AWS [9]. Além disso, o tempo de download menor agiliza o processo de busca de trechos no MAM.

Ao processo de entrada de mídias no MAM se dá o nome de *ingest* [7] (ingestão).

2.1.4 Pré-corte de mídias

Embora a capacidade de buscar e recuperar mídias de alta qualidade seja suficiente, em alguns casos, os materiais brutos, fontes para criação de conteúdo final, podem ser arquivos muito grandes (com alto volume de bits por tempo de vídeo, e.g. 50Mb por segundo) ou muito maiores que o trecho do vídeo que será usado na produção. Por isso, é possível marcar os vídeos de baixa qualidade com os pontos de corte e baixar posteriormente o material de alta qualidade desejado, já cortado.

2.2 Pesquisa sobre sistemas existentes no mercado

Conforme dito anteriormente, existem alguns sistemas de MAM no mercado que atendem, geralmente, a produção para TV. São eles: Media Central e Viz One. A seguir será apresentado brevemente cada um deles.

2.2.1 Media Central

A ferramenta Media Central da empresa Avid divide suas funcionalidades em diversos módulos, como os módulos de gerenciamento Editorial, de Produção, Redação e *Assets* [2]. Estes módulos oferecem suporte a workflows de *ingest* e publicação de conteúdo, busca, pré-corte e gerenciamento de acesso, como podemos ver na Figura 2.1.

Possui integração com produtos da própria Avid, como *playout*[1], editor de vídeo

[1] *Playout* é um tipo de sistema responsável pelo agendamento de vídeos que serão tocados.

Figura 2.1: Tela da ferramenta Media Central[1]

e gerenciamento/automação de estúdios jornalísticos [10], facilitando, assim, a personalização do fluxo de trabalho de produção como ilustrado na Figura 2.2. Conta ainda com diversas integrações com outras ferramentas e plataformas, como as de *storage* de arquivamento, transferência, transcodificação, controle de qualidade de mídia (QC) e publicação [11].

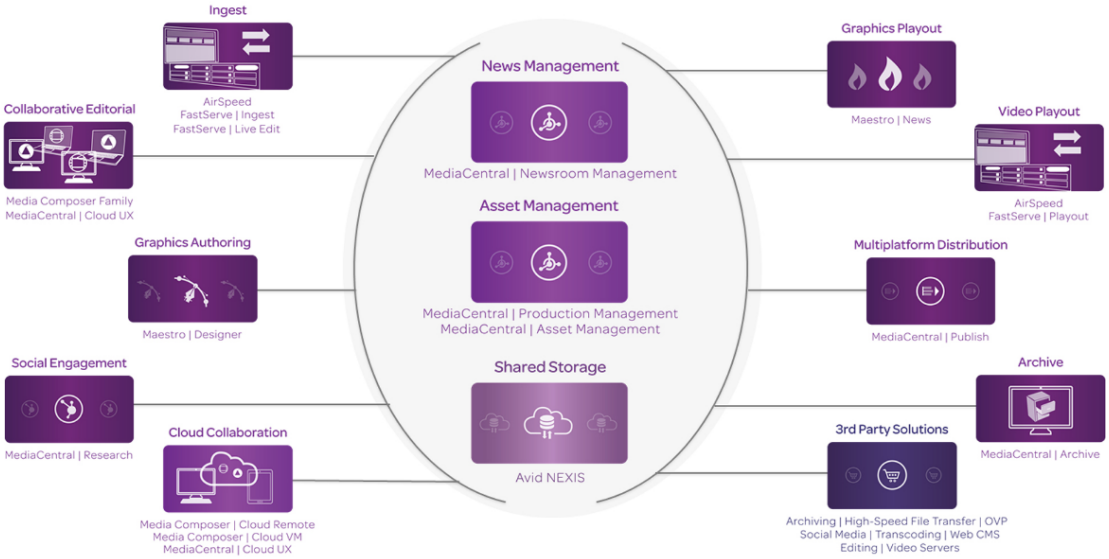

Figura 2.2: Fluxo de trabalho da ferramenta Media Central e suas Integrações [2]

2.2.2 Viz One

A solução de MAM da empresa Vizrt conta com os recursos básicos como ingestão e busca de conteúdo (como podemos ver na Figura 2.3, gerenciamento de acesso, normalização de conteúdo, geração de proxy e pré-corte [4]. Possui suporte a logging, configuração e customização de semântica de metadados, tornando as buscas

melhores e permitindo o ajuste do sistema a diferentes fluxos de trabalho, como ilustrado na Figura 2.4.

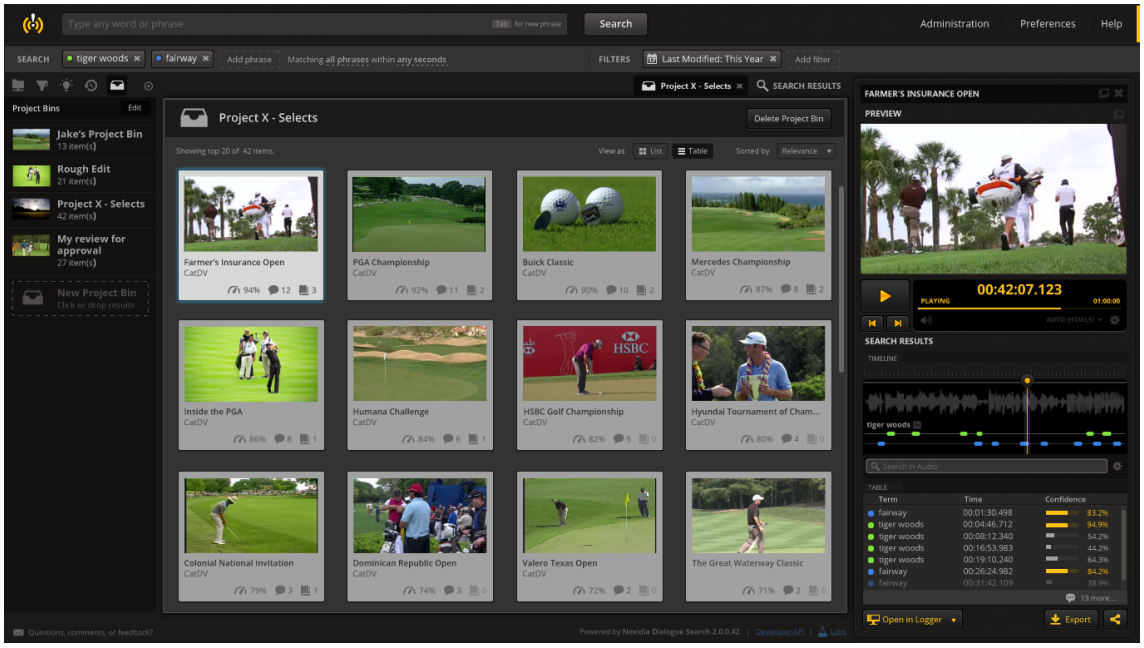

Figura 2.3: Tela da ferramenta Viz One [3]

O Viz One possui integração com outros produtos da mesma empresa, permitindo o gerenciamento de ativos gerados por suas ferramentas. Possui também integrações com ferramentas de outras empresas, como ferramentas de edição, sistemas de *playout*, controle de qualidade de mídia (*Quality Control - QC*) e motores de transcodificação. Expõe uma API, permitindo integrações/extensões de terceiros [4].

2.3 Sistema open source

Foi encontrado um único sistema para catalogação e acervo de mídia, o projeto Avalon [12]. Ele possui funcionalidades voltadas para a catalogação de vídeos, imagens e áudios, tais como *upload*, busca, controle de acesso e gerenciamento de *storage*. É usado em inúmeras universidades e bibliotecas.

Este, porém, não atende às necessidades de edição de conteúdo bruto presentes em um MAM. Sendo assim, não foi encontrada na pesquisa realizada nenhuma solução *open source* que atenda a demanda de gerenciamento de mídia para produção/publicação de conteúdo.

2.4 MXF

A SMPTE (*Society of Motion Picture and Television Engineers*) é uma associação técnica, fundada em 1916, responsável por inúmeras definições técnicas relacionadas a indústria audiovisual como o teste de colorbar, o timecode e o formato de vídeo MXF (*Material Exchange Format*)[13].

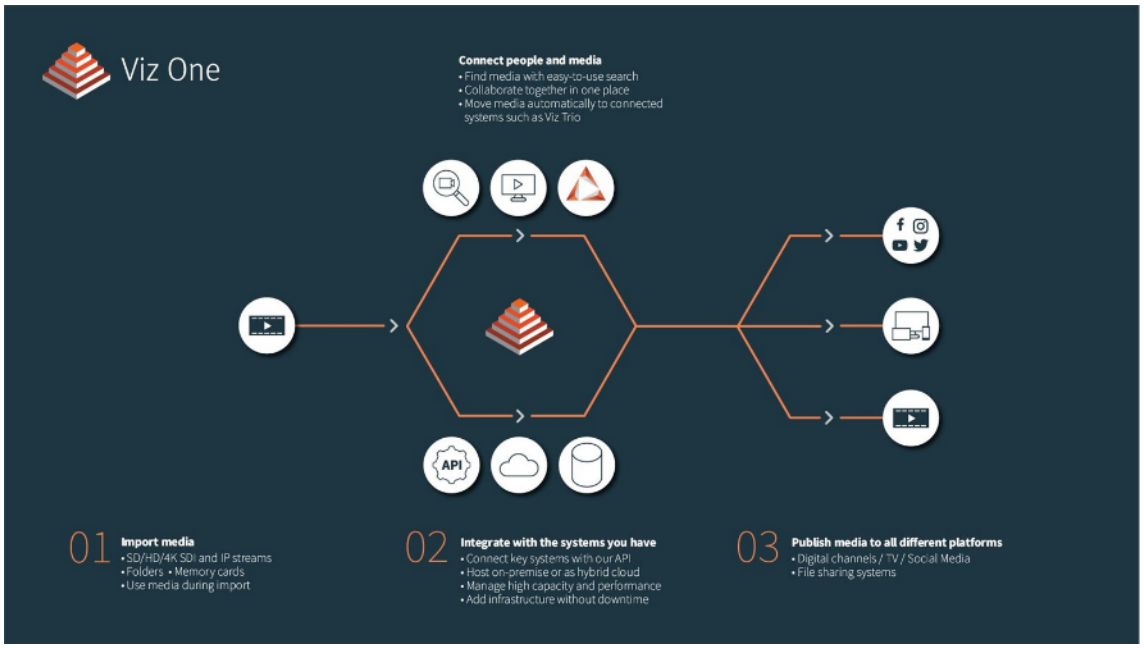

Figura 2.4: Fluxo de trabalho da ferramenta Viz One [4]

O formato MXF é, na verdade, um formato de contêiner onde se encontram trilhas de áudio, vídeo e dados (como `timecode` e *closed caption*)[14]. Na definição do formato, as trilhas de áudio e de vídeo são chamadas de essência e as trilhas contendo dados auxiliares são chamadas trilhas de metadados [7]. Ele não define uma codificação de audio ou vídeo específica, deixando para os fabricantes (de câmeras, editores, sistemas de gerenciamento e aparelhos de playback) as definições específicas de codificação, frequência de bits, etc. As trilhas de dados possuem um papel fundamental nos fluxos de trabalho da produção de conteúdo, pois permitem mapeamento e sincronização de eventos, como é o caso do timecode e acesso a dados de transcrição e descrição, providos pelo closed caption, que são muito importantes na produção de peças jornalísticas e esportivas.

Além dessas características, a indexação das essências confere ao MXF a vantagem de permitir o acesso incremental ao conteúdo, ou seja, é possível consumir uma mídia que ainda está sendo processada/codificada.

Apesar do formato de contêiner, dos *codecs* mais comuns (como MPEG-2 e MPEG-4) e dos formatos de interoperabilidade serem abertos, as combinações dos mesmos são de domínio de empresas fabricantes. Isso dificulta a implementação de ferramentas abertas ou de fácil acesso para pequenas organizações.

2.5 Considerações Finais

Neste capítulo, foram levantadas as principais funcionalidades e práticas dos sistemas de MAM presentes no mercado (na Seção 2.1) além das características específicas de cada sistema (na Seção 2.2).

Na Seção 2.3 foi realizada uma pesquisa sobre os sistemas de MAM de código aberto, onde foi revelado que o único sistema encontrado não atende as funcionalidades levantadas na Seção 2.1, corroborando com a necessidade de criação de um sistema de MAM *Open Source*.

O levantamento das características e funcionalidades proporcionaram uma aproximação mais concreta da definição do MAM. De posse dessas informações, é possível estabelecer as funcionalidades e características mínimas e as desejadas para a implementação do MAM aberto. A definição do sistema desenvolvido e suas características são apresentadas no capítulo a seguir.

Capítulo 3

Tecnologias utilizadas

Neste capítulo são descritas as tecnologias usadas para o desenvolvimento do uMAM

3.1 Python

A quarta linguagem de programação mais usada no mundo, e a com maior crescimento, segundo o índice TIOBE 2018, Python, foi criada em 1989 por Guido von Rossum, baseada na linguagem ABC [15]. Esta popularidade se deve, em parte, à ubiquidade da linguagem nos campos relacionados à computação como estatística, ciência de dados, aprendizado de máquina, administração de sistemas, desenvolvimento web, etc.

Python é uma linguagem interpretada, com tipagem forte e dinâmica. Tem foco na legibilidade, se aproximando bastante da linguagem natural (inglês) e implementa múltiplos paradigmas, como Orientação a Objetos, funcional e procedural. Além disso, possui uma biblioteca padrão extensa e estável, permitindo o desenvolvimento rápido e preciso.

Visto que a linguagem é amplamente utilizada, com adoção crescente, de fácil utilização e rico ecossistema, podemos dizer que o seu uso está em conformidade com os princípios de facilidade de contribuição e extensibilidade. Portanto, Python foi escolhida para ser a linguagem utilizada nos componentes centrais da lógica de negócio do projeto.

3.2 Rust

Criada pelo time de pesquisa da Mozilla, *Mozilla Research*, afim de permitir o desenvolvimento seguro de aplicações de alta performance, diminuindo a distância entre o desenvolvimento de baixo nível e o gerenciamento seguro de memória.

Rust é, então, uma linguagem de baixo nível, com gerenciamento estático de memória (checado em tempo de compilação), sem *garbage collection* que implementa alguns paradigmas como procedural e funcional. Sua biblioteca padrão é pequena, se comparada com outras linguagens mais populares e de alto nível, porém possui uma extensa gama de módulos para permitir o desenvolvimento seguro e eficiente, mesmo em casos de uso de concorrência e paralelismo.

Dadas tais características, a linguagem foi escolhida para o desenvolvimento de componentes que necessitam de alta performance, devido à sua alta carga de

processamento e I/O. Esta decisão foi tomada a fim de atender o princípio da solução de promover agilidade, citado na Seção 4.1.3.

3.3 Flask

Flask é um *microframework* web escrito em Python que auxilia na criação de aplicações web . Ele é desenhado de maneira minimalista, não possuindo uma estrutura padrão de aplicação ou camadas específicas, como camadas de modelo ou visual [16].

Este minimalismo favorece a criação de microsserviços, permitindo que os mesmos possuam a estrutura mínima para o seu funcionamento. Além de seu minimalismo, o seu vasto ecossistema também garante que diversas funcionalidades possam ser implementadas de maneira ágil e de fácil manutenção.

3.4 SQLAlchemy e Alembic

SQLAchemy é uma poderosa biblioteca de interface com banco de dados e ORM escrita em Python. Ela possui adaptadores para diversos SGBDs como MySQL, Postgres, MariaDB, SQLServer, SQLite, OracleDB, DB2, provendo interface para a maior parte dos idiomas de SQL, embora permita escrita de *queries* brutas e parametrizadas.

Alembic é uma biblioteca e ferramenta que cuida de migrações de modelos de banco de dados, utilizando a camada de banco de dados do SQLAlchemy. De posse de um conjunto de modelos do SQLALchemy, ela é capaz, ainda, de gerar *scripts* de migração automaticamente, facilitando a integração entre desenvolvimento e administração da base de dados da aplicação.

3.5 Ansible

Ansible é uma ferramenta de código aberto desenvolvida na linguagem Python que tem por objetivo tornar a execução de tarefas de administração de sistemas e provisionamento de infraestrutura até o nível do sistema operacional fácil e reprodutível [17]. A ferramenta possui alguns conceitos chave:

- *Host* - um servidor, identificado por seu endereço
- *Group* - agrupamento de *Hosts* ou de *Groups*
- Roles - conjunto de características parametrizadas a serem aplicadas a um *Group* ou *Host*
- *Variable* - Um parâmetro a ser aplicado em uma Role
- *Inventory* - Um conjunto de *Hosts*, *Groups* e *Variables*
- Playbook - Mapeamento entre Groups e Roles

A partir destes conceitos é possível criar projetos de implantação e administração de sistemas de maneira modular, permitindo a composição e reúso de Roles. De fato,

a ferramenta possui um gerenciador de pacotes, o Ansible Galaxy, que permite o uso de Roles de terceiros em um projeto.

Um projeto Ansible é, basicamente, composto por um conjunto de roles e um conjunto de *Inventories*. Cada *Inventory*, normalmente, representa um ambiente de implantação, e.g. ambientes de desenvolvimento, homologação e produção, tornando o mapeamento de elementos do sistema mais visíveis.

3.6 FFmpeg

FFmpeg é o nome de uma biblioteca e uma ferramenta usadas para codificar, decodificar, transcodificar, analisar e tocar arquivos de áudio e vídeo [18]. Ela suporta diversos formatos de codificação de áudio e vídeo e contêineres. Suas funcionalidades são expostas por 3 aplicações: ffmpeg (transcodificação), ffplay (reprodução) e ffprobe (análise).

O uso desta ferramenta é essencial para realizar a normalização dos vídeos que entram e que saem da plataforma, pois permite que os vídeos usados estejam num formato apropriado para o fluxo de trabalho dos usuários.

O FFmpeg é distribuído sob a licensa GPLv2 [19], permitindo que o uso e a distribuição do uMAM assuma formatos comerciais e não-comerciais, sem nenhum prejuízo.

3.7 RabbitMQ

RabbitMQ é um software de mensageria (*message broker*) de código aberto, desenvolvido na linguagem Erlang, que implementa entre outros protocolos o AMQP 0-9-1. Seu funcionamento e utilização se estruturam em torno de alguns conceitos. São eles:

3.7.1 Produtor

É o nó cliente (externo ao *broker*) que produz mensagens. Uma mensagem enviada por um produtor é composta de:

- *Exchange*: A *exchange* onde será publicada a mensagem
- *Routing key*: A chave de roteamento da mensagem a ser usada pela *exchange*
- *Payload*: O corpo da mensagem
- *Headers*: Metadados adicionais da mensagem na forma de chave-valor.

3.7.2 *Exchange*

É um componente interno do *broker* que funciona como um barramento e é responsável pelo roteamento de mensagens enviadas pelos consumidores a serem entregues nas filas. Existem 4 tipos de roteamento no RabbitMQ:

- *Direct*: Envia a mensagem para todos os *bindings* que possuem o *routing key* exatamente igual ao da mensagem.

- Topic: No modelo de tópicos, a *routing key* da mensagem deve ser formada por uma lista de palavras separadas por pontos enquanto a *routing key* do *binding* deve ser formada por um padrão de nome onde o caractere '*' representa uma e somente uma palavra qualquer e o caractere '#' representa nenhuma ou muitas palavras quaisquer.
- Fan-out: Envia a mensagem para todos os *bindings*, independentemente da *routing key* da mensagem ou dos *bindings*.
- Envia a mensagem para os *bindings* baseado na comparação entre os valores dos campos dos *headers* das mensagens e os argumentos dos *bindings*.

3.7.3 Fila

É um componente interno do *broker* que é responsável pelo armazenamento das mensagens a serem consumidas no modelo FIFO (*First in, First out*). As mensagens chegam nas filas através de *bindings* entre elas e as *exchanges*. Uma fila pode ser persistente ou transitória, sendo escrita em disco, no primeiro caso ou apenas em memória, no segundo.

3.7.4 Binding

É um componente interno do *broker* que representa uma relação de consumo entre uma fila e um *exchange* ou entre dois exchanges. Um *binding* possui os seguintes componentes:

- Origem: *Exchange* de origem das mensagens
- Destino: *Exchange* ou fila para onde serão roteadas as mensagens
- *Routing key*: A chave de roteamento usada para definir se a mensagem será ou não enviada para o destino, baseada no tipo da origem.
- *Arguments*: metadados adicionais dos *bindings* usados, normalmente, quando a origem é do tipo *headers*.

3.7.5 Consumidor

É o nó cliente (externo ao *broker*) que consome as mensagens de uma fila através da abertura de uma conexão.

3.8 Considerações Finais

Neste capítulo, foram descritas algumas das principais tecnologias (bibliotecas, plataformas e *frameworks*) utilizadas no desenvolvimento e operação do sistema proposto para este trabalho. Cada tecnologia tem seu papel habilitador nos diversos aspectos do sistema, como podemos ver na tabela 3.8.

O uso da linguagem Python com Flask, SQLAlchemy e Alembic permitiram a criação mais rápida, precisa e gerenciada dos serviços do sistema. O uso da linguagem Rust permitiu o desenvolvimento de componentes com maior performance, sem

Tecnologia	Descrição	Papel
Python	Linguagem de programação dinâmica e interpretada	Desenvolvimento dos principais componentes
Rust	Linguagem de programação performática, segura e expressiva	Desenvolvimento de componentes de alta performance
Flask	*Microframework web* escrito em Python	Desenvolvimento de componentes que se comunicam via HTTP
SQLAlchemy	Biblioteca de interface com bancos de dados e ORM escrita em Python	Definição dos modelos de dados dos serviços do sistema
Alembic	Ferramenta de migrações de banco de dados utilizando SQLAlchemy	Facilitar a operação do sistema integrado aos modelos de dados dos serviços
Ansible	Ferramenta de automação de operações de sistemas	Automação de *deploys* e provisionamento de infraestrutura
FFmpeg	Ferramenta de transcodificação de audio e vídeo	Auxiliar as tarefas de manipulação de arquivos de vídeo
RabbitMQ	Software de mensageria baseado no protocolo AMQP	Auxiliar o gerenciamento de tarefas assíncronas no sistema

Tabela 3.1: Resumo das tecnologias utilizadas e seu papel no desenvolvimento e operação do uMAM

grande perda de expressividade na escrita e com forte apoio ferramental. A escolha do *broker* RabbitMQ, para gerenciamento de tarefas assíncronas, e da ferramenta FFmpeg, para transcodificação de vídeos, auxiliaram na criação de *workflows* de mídia. Por fim, o uso da ferramenta ansible auxiliou na diminuição das lacunas entre desenvolvimento dos componentes e a implantação do sistema, permitindo o crescimento do mesmo, durante o seu desenvolvimento.

Capítulo 4

Solução proposta

A solução proposta e desenvolvida para este trabalho se refere a um MAM voltado para produtores independentes, o uMAM. Ou seja, não tem como objetivo atender grandes organizações ou corporações, refletindo as características distribuídas das novas formas de produção de conteúdo.

4.1 Premissas

As premissas da solução são aquelas que guiam o desenvolvimento e as decisões de adição ou não de funcionalidades na solução proposta. São elas: baixo custo de manutenção, garantia de qualidade de mídia, agilidade e acessibilidade.

4.1.1 Baixo custo de manutenção

A solução proposta tem como objetivo melhorar a qualidade e produtividade de pequenas e médias organizações produtoras de conteúdo. Desta forma, se faz necessário o baixo custo de manutenção do sistema, afim de viabilizar a sua utilização quando há menor presença de recursos econômicos/financeiros.

4.1.2 Garantia de qualidade da mídia

A garantia de qualidade da mídia é uma das características mais importantes em um MAM. Ao garantir que as mídias estão normalizadas e em boas condições de consumo, o sistema diminui a necessidade de intervenção manual para correção de conteúdos, permite maior agilidade no processo de produção e melhora a qualidade do produto final.

4.1.3 Agilidade

A fim de garantir um processo de produção mais veloz, o próprio sistema deve ser ágil. Ele deve, também, permitir o acompanhamento e visualização de status de seus processos. A ausência de acompanhamento de processos obriga o usuário a parar o seu fluxo de trabalho, perdendo agilidade.

4.1.4 Acessibilidade

O sucesso de uma solução de código aberto depende, principalmente, da adoção e engajamento dos usuários e contribuidores. Com o objetivo de garantir esta adoção e engajamento, a solução possui a premissa de tentar garantir a possibilidade de uso pela maior parte dos seus candidatos a usuário.

4.2 Fluxo de Trabalho

O fluxo de trabalho no uMAM pode ser dividido em 3 partes: *Ingest*, Curadoria e Exportação.

4.2.1 *Ingest*

É o nome dado ao processo de entrada de vídeos no uMAM. Este processo pode ser realizado via *upload* ou integração com outros sistemas.

4.2.2 Curadoria

É o nome dado ao processo de busca de vídeos, escolha de trechos dos mesmos e adição dos trechos à listas de edição.

4.2.3 Exportação

É o processo de corte dos trechos marcados nos vídeos nas listas de edição e disponibilização para o usuário. Pode ser realizado via *download* ou integração com outros sistemas.

4.3 Funcionalidades

Baseado no estudo das funcionalidades exposto na Seção 2.1 foram levantadas as funcionalidades do sistema que serão descritas a seguir. As funcionalidades propostas para a solução visam permitir e melhorar o fluxo de produção de conteúdo. Elas também devem sempre respeitar as premissas propostas para a solução.

4.3.1 Autenticação de usuários

Como se trata de armazenamento de material não publicado, se faz necessário o controle mínimo de acesso aos conteúdos. Este controle é feito através de identificação de um usuário cadastrado e validação de sua senha (Figura 4.1).

4.3.2 Entrada de mídias

O usuário deve conseguir fazer envio de mídias e seus metadados para o sistema. Para atingir tal objetivo, o sistema dispõe de uma interface de *upload* de mídias. Há também, uma interface programática (API) afim de oferecer entrada de conteúdo de outros sistemas ou interfaces de terceiros.

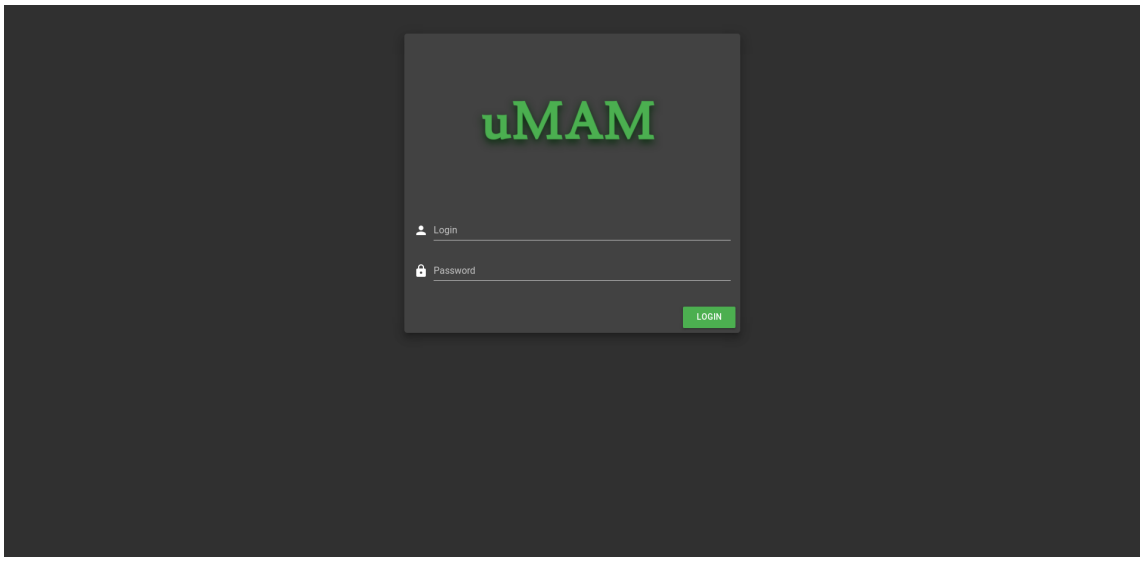

Figura 4.1: Tela de *Login*

Esta funcionalidade pode ser útil em situações de colaboração em diferentes posições geográficas. Por exemplo, numa gravação de um fato ou cena, os vídeos gravados podem ser enviados diretamente para os sistema, para uso.

4.3.3 Busca de mídia e indexação de metadados

Para permitir maior agilidade na obtenção de conteúdos, um MAM conta com uma interface de busca (Figura 4.2) e processos de indexação de metadados. Quanto melhor o processo de indexação, mais úteis serão os resultados apresentados.

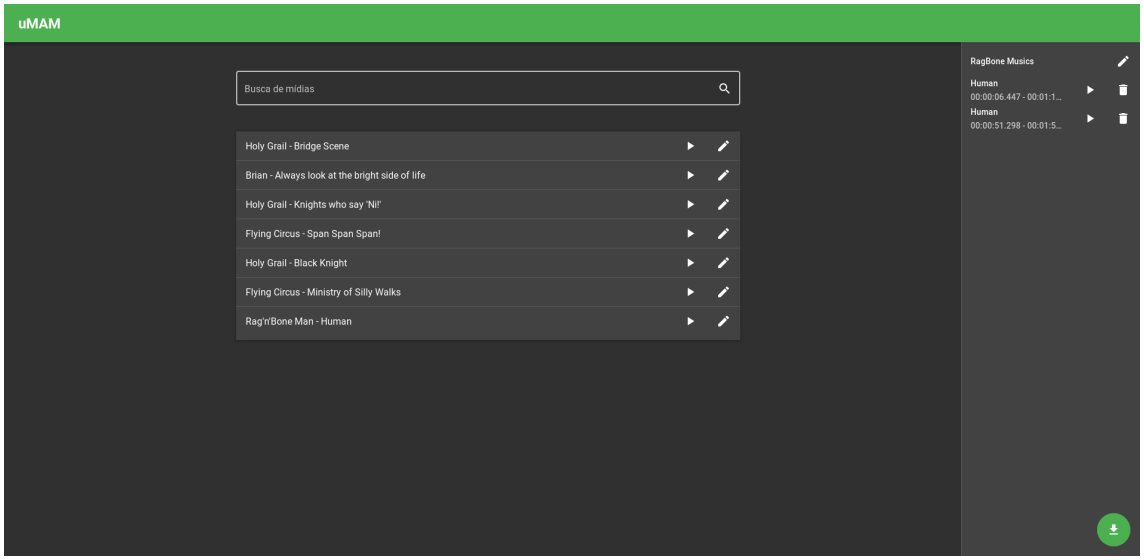

Figura 4.2: Tela de busca de *assets*

4.3.4 Pré-visualização de mídia em baixa resolução

Além da possibilidade de localizar conteúdos, o sistema possui a capacidade de tocar versões de baixa resolução dos vídeos armazenados. Esta funcionalidade é importante, pois impede o tráfego de vídeos de alta resolução, diminuindo o consumo de banda de rede e garantindo o baixo custo de manutenção.

4.3.5 Listas de trabalho

Durante o processo de obtenção de materiais brutos para edição, se faz necessária a marcação de trechos de vídeos e sua catalogação. Para este fim, existem as listas de edição, que permitem a seleção e compartilhamento de materiais pesquisados e catalogados. Um jornalista pode, por exemplo, iniciar a marcação de trechos de vídeos gravados para uma matéria jornalística e salvá-la para baixar quando todos os trechos forem marcados. No uMAM, somente o usuário que realizou a criação da lista pode altera-la.

4.3.6 Pré-corte e download de alta resolução

A fim de continuar o trabalho com os conteúdos selecionados, o sistema permite a obtenção dos mesmos, através de corte do conteúdo de alta resolução e posterior disponibilização. Para tal, o sistema disponibiliza um editor simples de vídeo como mostra a Figura 4.3, onde são marcados os pontos de início e fim dos trechos a serem adicionados a uma lista de edição (chamados de clipes) e exportados. A lista pode, então, ser exportada (com os cortes feitos em alta resolução) e exposta para download. Os processos de corte realizados podem ser acompanhados na tela de acompanhamento, como mostra a Figura 4.4

Um exemplo de utilização desta funcionalidade é o uso de trechos marcados numa lista de edição para produção de uma matéria jornalística.

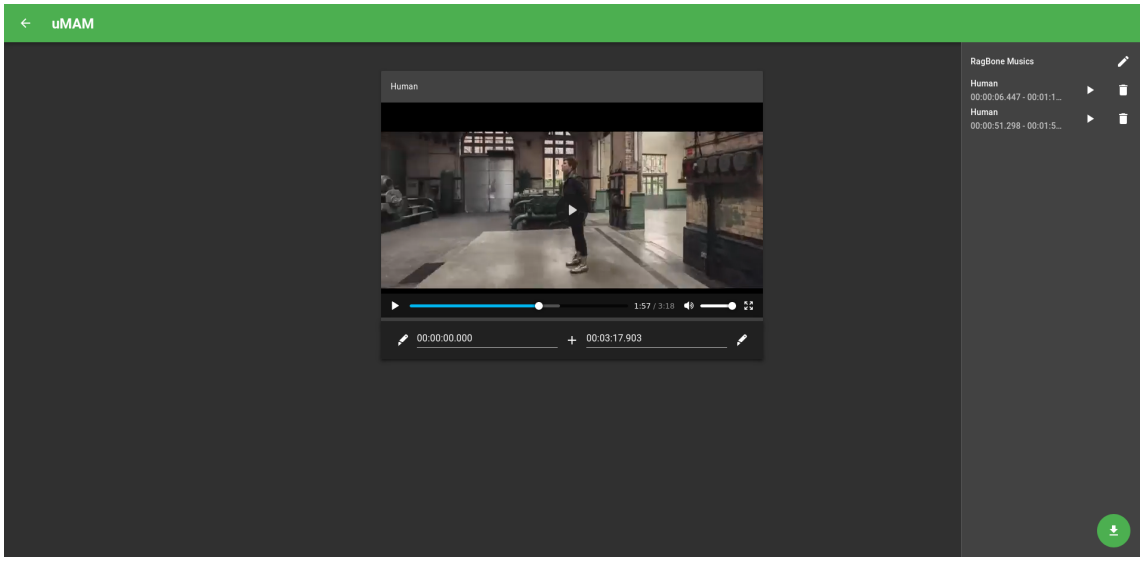

Figura 4.3: Tela de edição

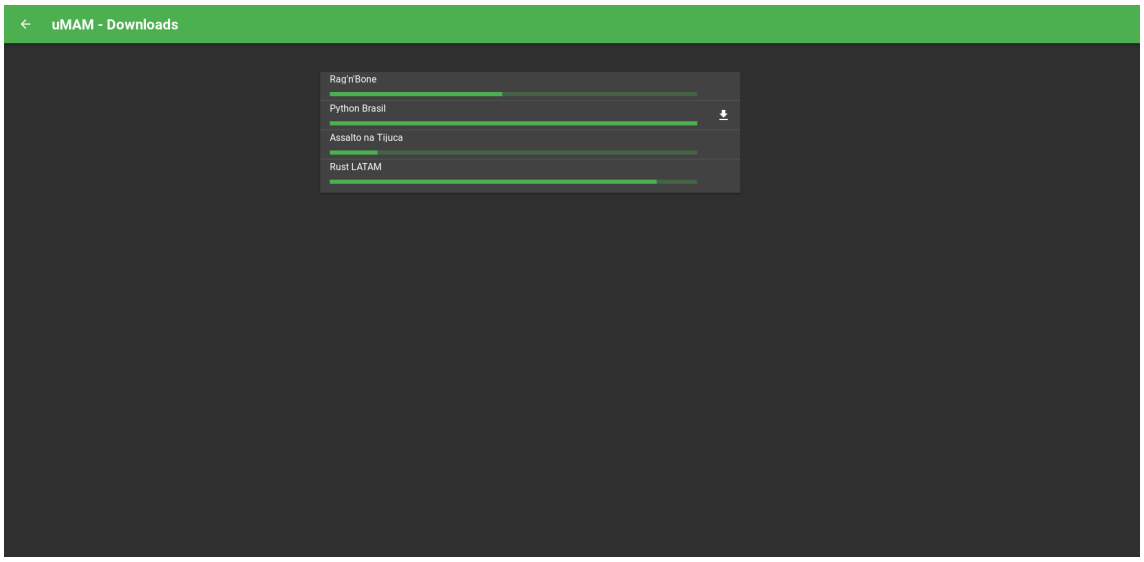

Figura 4.4: Tela de acompanhamento de requisições de pré-corte

4.3.7 Exportação

O uMAM dispõe de uma interface genérica de publicação, através do protocolo SFTP configurável, a fim de suprir necessidades básicas de integração com outros sistemas.

Caso o usuário trabalhe como um gerador de conteúdo, e não um distribuidor/exibidor (como é o caso de uma agência de notícias), esta funcionalidade pode ser usada para enviar o conteúdo de uma lista de edição para os distribuidores/exibidores.

4.4 Formato da mídia

Visando garantir maior acessibilidade e considerando os alvos de publicação de conteúdo, o formato de vídeo escolhido foi o MP4, tanto para a alta resolução quanto para a baixa resolução. Apesar de ser um formato menos poderoso em termos de rastreamento de dados e agilidade de edição, é o formato com maior compatibilidade em navegadores e aparelhos fim, atendendo as premissas de baixo custo de manutenção e acessibilidade.

4.5 Considerações finais

Neste capítulo pudemos, de maneira mais sólida, visualizar a ferramenta que o usuário do uMAM terá a sua disposição. Na Seção 4.1, foram definidas as premissas nas quais a solução se baseia. Na Seção 4.3 foram propostas, com base nas premissas e na pesquisa realizada no Capítulo 2, as principais funcionalidades do sistema.

Estas definições são fundamentais para o desenho e desenvolvimento do sistema, sempre guiados pelas funcionalidades levantadas e pelas premissas definidas.

Capítulo 5

Desenho da Solução

Neste capítulo é apresentada uma visão ampla do desenvolvimento do sistema proposto no Capítulo 4. Na Seção 5.1, são definidas algumas premissas que nortearão as decisões de desenho, implementação e escolha de tecnologia no projeto. A seguir, na Seção 5.2, são detalhados os desenhos arquiteturais dos principais componentes do sistema, o MAM, onde são implementadas as funcionalidades relacionadas a recuperação e gerenciamento de mídias, e o sistema de *Workflow* onde ocorrem as tarefas assíncronas relacionadas a manipulação de vídeos e arquivos.

5.1 Premissas do projeto

Antes de detalhar as características arquiteturais do projeto, serão definidas algumas premissas que guiarão, junto com as premissas da solução (levantadas no Capítulo 4), as decisões de implementação do projeto.

5.1.1 Facilidade de contribuição

Para permitir o sucesso e manutenção de um projeto *open source*, é necessária a existência de uma comunidade forte de usuários e contribuidores. Para isso, as decisões de implementação, desenho e documentação devem favorecer a colaboração. Por este motivo, facilidade de contribuição é uma das premissas do projeto.

5.1.2 Extensibilidade

Um sistema de MAM, embora possua papel central no fluxo de produção de conteúdo audiovisual, não cobre todas as etapas do processo, como distribuição, arquivamento, captação e edição. Para que seja possível a interação com sistemas que cumprem estes e outros papéis no fluxo de produção de conteúdo e adaptar seu uso a fluxos de trabalho específicos, o sistema deve ser capaz de permitir extensões e customizações de maneira natural.

5.2 Arquitetura

O sistema se divide em dois grandes componentes: o MAM, peça central no gerenciamento de mídias; o Sistema de *Workflow*, responsável pelas tarefas assíncronas e agendadas a serem realizadas de maneira desacoplada ao gerenciamento das mídias.

O sistema deve contar, ainda, com repositórios de arquivos para armazenar as mídias, estes repositórios não fazem parte do escopo de desenvolvimento do projeto, sendo, portanto, escolhida uma opção de sistema de *storage* distribuída, ou não, levantada no Capítulo 7.

5.2.1 MAM

O componente de MAM é responsável pelo gerenciamento dos ativos e por todas as interações do usuário com o sistema. Ele foi estruturado numa arquitetura de microsserviços, a fim de facilitar sua extensibilidade e escalabilidade.

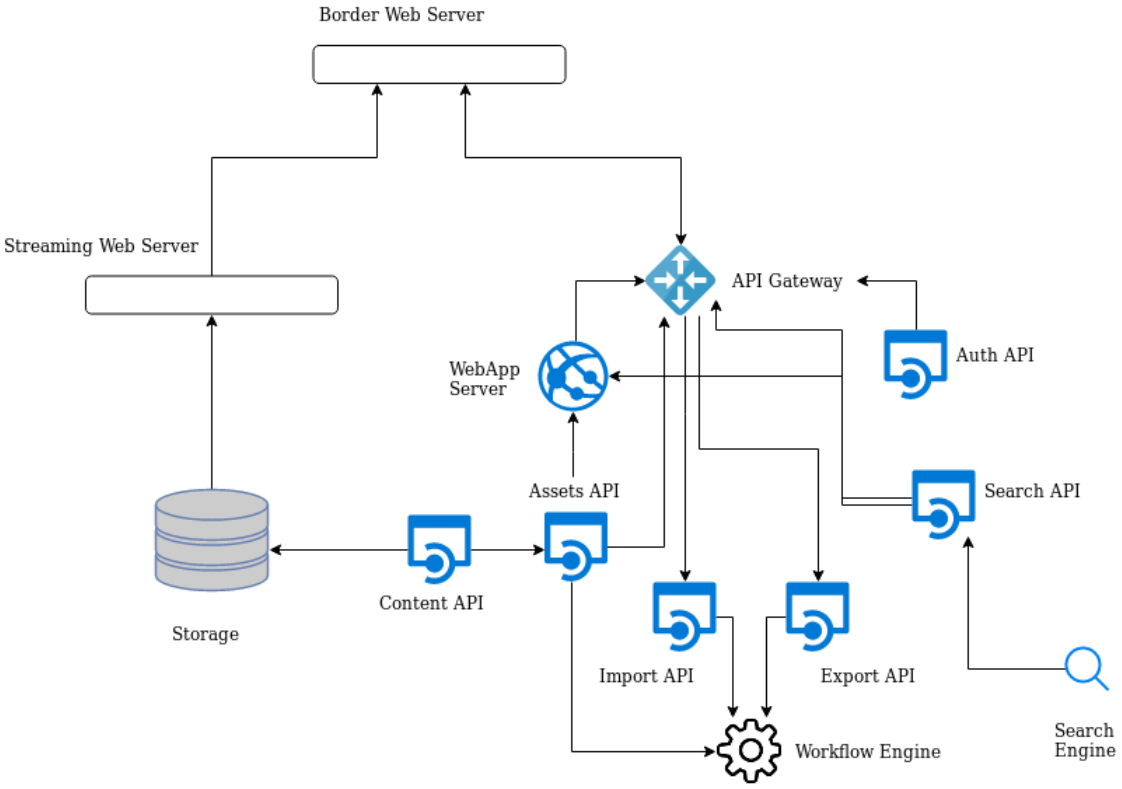

Figura 5.1: Arquitetura do Sistema MAM

Esse sistema é composto de um servidor web com informações persistidas em um banco de dados relacional e um motor de busca, onde serão indexados os metadados a serem buscados. Foi desenvolvido um cliente web para a interação com o usuário.

Border Web Server (BWS)

Servidor web mais externo à aplicação. Seu objetivo é servir conteúdo http de maneira eficiente e de forma a reduzir o máximo possível a carga dos componentes mais internos através de cache.

Este componente não foi desenvolvido para este projeto, sendo usado um *web server* já existente como o NGINX. Ele pode ser fundido com o componente de *API Gateway*, dependendo do modelo de implantação.

Content Web Server (CWS)

De maneira semelhante ao BWS, tem por objetivo servir conteúdo via HTTP através de um software de *web server*. Neste caso porém, o propósito é servir conteúdo de vídeo e imagens disponibilizados para o usuário consumir.

O CWS possui configurações específicas para permitir a reprodução de vídeos e *download* de conteúdo. O conteúdo é servido diretamente dos *storages* usados, sendo necessária a configuração do ambiente para permitir a acessibilidade a estes conteúdos.

Content API

Para garantir a segurança de armazenamento nos *storages*, nenhum conteúdo fica disponível para o CWS. O mapeamento publico, visível para o CWS é, então, preenchido por *links* simbólicos para os caminhos dos respectivos arquivos.

Uma tarefa agendada no sistema de *workflow* garante que estes *links* serão removidos após o tempo de expiração.

API Gateway

Este componente representa um *proxy* de acesso para todos os serviços do sistema. Ele é responsável pelo balanceamento de carga e contribui para a separação de responsabilidades dos serviços.

Toda requisição direcionada a API do uMAM, é enviada para a Auth API para verificação de autenticação/sessão. O mecanismo de autenticação é explicado a seguir.

Auth API

Serviço responsável pela autenticação e verificação de sessões. Este serviço possui uma base de dados relacional, no caso da implementação base, MySQL, onde ficam armazenadas informações dos usuários necessárias para sua autenticação e uma base não relacional de chave/valor para armazenamento de sessões, no caso, Redis.

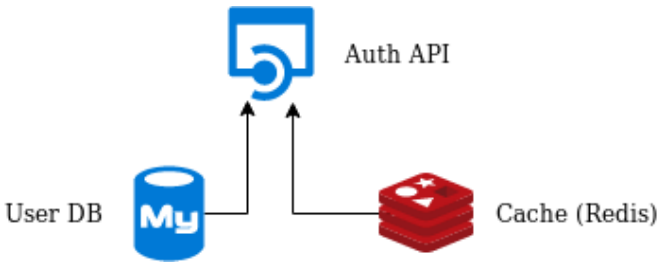

Figura 5.2: Arquitetura do Serviço Auth API

A *Auth API* pode ser acionada, então, para três fluxos. Para *login*, neste caso ela valida as credenciais do usuário e cria uma sessão, retornando a chave da sessão para uso pelo cliente. Para *logoff*, neste caso ela remove uma entrada de uma chave existente.

O terceiro caso é o de validação de sessão. Neste caso ela, ou retorna as informações de usuário no cabeçalho da requisição, caso uma chave de sessão exista,

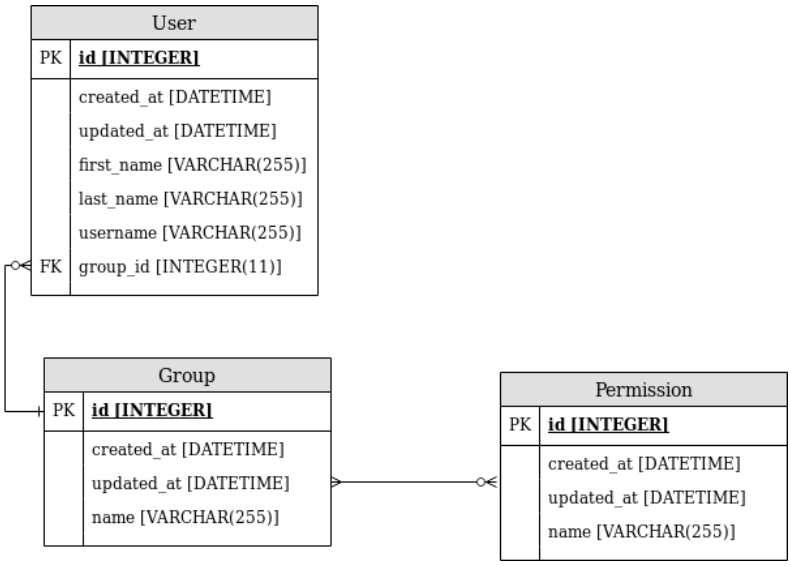

Figura 5.3: Diagrama de Entidade Relacionamento - Auth API

ou retorna um código de *status* não autorizado (401) caso a sessão não exista. As informações retornadas são, então, repassadas pelo *API gateway* às APIs para uso nos outros serviços.

A *Auth API* expõe seu serviço através de uma API REST. Ela foi desenvolvida na linguagem Python usando o *framework* Flask para a interface HTTP e o ORM SQLAlchemy para mapeamento do modelo de dados. A aplicação segue o modelo arquitetural MVC e possui um *endpoint* para cada fluxo.

Assets API

É o serviço central para a lógica de negócio do sistema. É responsável por gerenciar metadados dos ativos e os apontamentos de suas versões de mídia, as listas de edição e seus *clips* e os perfis de codificação e empacotamento. Sendo assim, é na *Assets API* que ocorrem os seguintes fluxos:

- Inserção, atualização e deleção de ativos
- Recuperação de dados completos de um *asset*
- Inserção, recuperação, atualização e deleção de listas de edição e clips
- Inserção, recuperação, atualização e deleção de perfis de codificação e empacotamento

É importante ressaltar que não é responsabilidade da *Assets API* a busca ou indexação de ativos e listas de edição. A responsabilidade da *Assets API* é de enviar, a cada operação de dados sobre ativos ou listas de edição, uma mensagem para o sistema de *workflow* para que a operação seja indexada, tornando, assim, o processo de busca e indexação, desacoplado da gravação/recuperação de dados.

A *Assets API* expõe seu serviço através de uma *API REST*. Assim como a *Auth API*, foi desenvolvida na linguagem Python usando o *framework* Flask para a interface HTTP e o ORM SQLAlchemy para mapeamento do modelo de dados. A aplicação segue o modelo arquitetural MVC e possui *endpoints* específicos para cada recurso (*assets*, *edit-lists* e *profiles*).

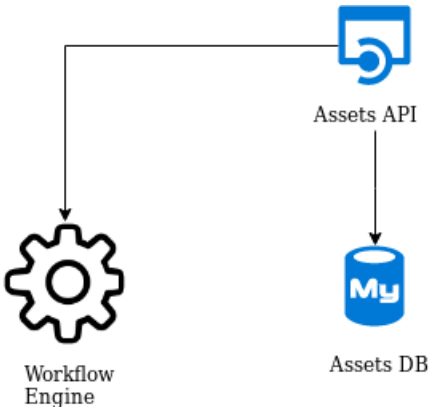

Figura 5.4: Arquitetura do Serviço Assets API

Search API

A *Search API* oferece a interface de busca de *assets* e listas de edição. Para prover este serviço, a *Search API* se conecta a um motor de busca, o ElasticSearch. Embora seu desenho seja bem simples, como observado na Figura 5.6, ela abstrai o processo de busca, permitindo a adoção de outras ferramentas e técnicas de indexação de metadados e busca.

Web App Server (WAS)

O *Web App Server* é o componente que centraliza todas as interações de interface de usuário. Ele o faz utilizando o Nuxt.JS [20], uma biblioteca que permite a criação de *webapps* universais (ou isomorficas), ou seja, aplicações web ricas no *frontend* que possuem alguma renderização dinâmica no *backend* [21], trazendo assim o dinamismo de uma aplicação de uma única página (*single page application*, SPA) e a velocidade da renderização no *backend*. Todas as páginas servidas pelo WAS devem passar por verificação de autenticação do usuário, a exceção da página de autenticação (*login*).

Import API

Import API é um serviço cuja função é gerenciar a criação de processos de entrada de mídias no MAM. A existência deste serviço abstrai o *workflow* de entrada, permitindo, assim, o uso de outros sistemas de *workflow* mais simples ou mais complexos, de acordo com as necessidades de uso do sistema.

Export API

O *Export API* gerencia os processos de exportação de conteúdo. A forma mais simples de exportação é aquela para o cliente pela exposição para *download*. É possível também a exportação para outro sistema ou plataforma, seja pelo simples envio para outro *storage* através de protocolo de transferência de arquivo como FTP ou SFTP, seja pela publicação em plataformas como Youtube, Twitter e Facebook.

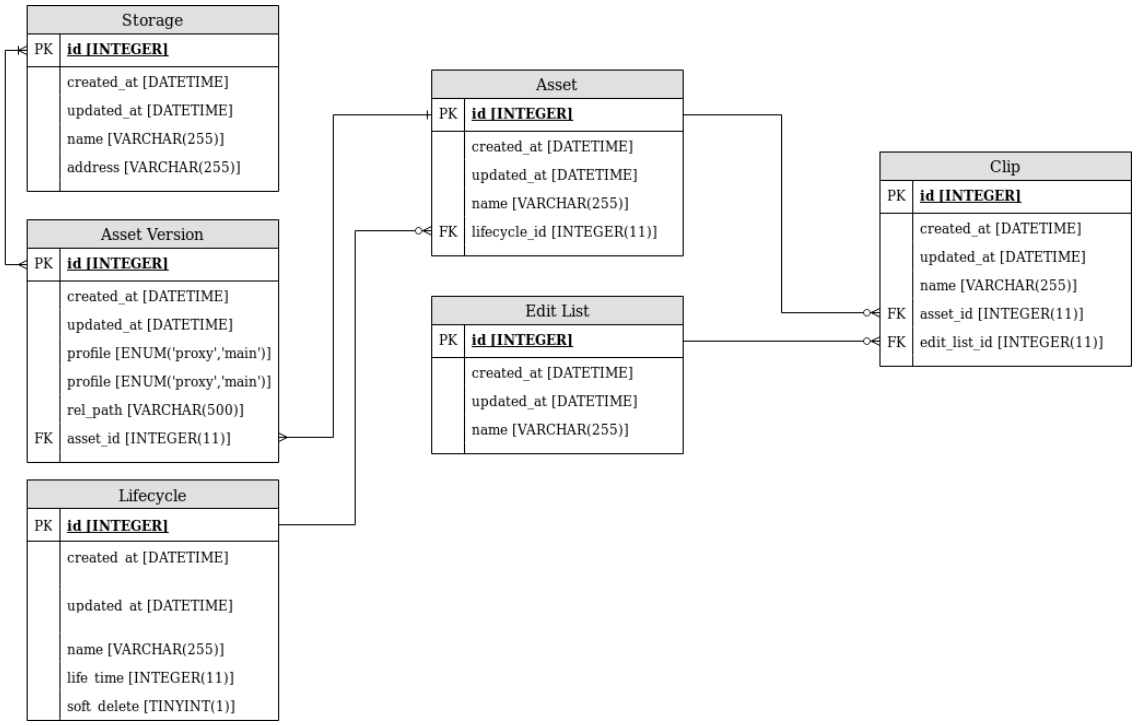

Figura 5.5: Diagrama de Entidade Relacionamento - Assets API

5.2.2 *Workflow*

O sistema de *workflow* desenvolvido, exclusivamente para o uMAM, tem como objetivo atender as demandas de fluxos assíncronos de processamento e transferência de mídias (entrada, corte e saída). Para tal propósito, ele dispõe de um servidor web que expõe uma API REST (para realizar o gerenciamento das tarefas a serem realizadas), um SGBD (responsável pela persistência dos estados das tarefas), um sistema de gerenciamento de mensagens e filas (para permitir a comunicação assíncrona), um *software* de *watchfolder* (responsável pelo recebimento dos arquivos e gatilho das tarefas), um *software* adaptador das mensagens de resposta e *softwares* responsáveis pela realização das tarefas em si. Como podemos ver na Figura 5.7, existem componentes arquiteturais representados por pastas. Estas pastas são, usualmente, pastas compartilhadas na rede via montagem no sistema de arquivos (e.g. via NFS), ou seja, as transferências de arquivo são transparentes para os *softwares* que as utilizam, via sistema de arquivos.

Neste sistema de *workflow*, os *workers* são peças fundamentais no funcionamento. Isto ocorre pois os fluxos são definidos de maneira linear (sem ramificações) dentro dos *workers* e qualquer ramificação ocorre via evento de redirecionamento, terminando o trabalho naquele *worker*. Este *framework* foi implementado para este trabalho na biblioteca "bureaucrat". O funcionamento da biblioteca é detalhado na Seção 6.4. Neste trabalho foram implementados os *workers*: Entrada, Pré-corte e quarentena.

Servidor *web (api)*

Desenvolvido na linguagem Python, utilizando a biblioteca Flask (HTTP), é o componente que inicia, busca e gerencia tarefas. Para persistir os estados das tarefas, os

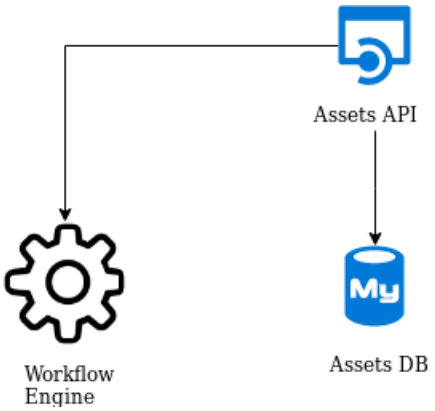

Figura 5.6: Arquitetura do Serviço Search API

roteamentos de fluxo, foi utilizado o banco de dados relacional MySQL através da biblioteca de ORM SQLAlchemy. Para enfileirar e rotear as tarefas para seus respectivos realizadores ("*workers*"), foi utilizado o *software* de mensageria RabbitMQ com o protocolo AMQP 0.9.1 através da biblioteca pika.

Foi utilizada, ainda, a ferramenta Alembic para gerenciar e versionar migrações do banco de dados.

Watchfolders

Estes são *softwares* responsáveis por aguardar a escrita de uma mídia (e, possivelmente, um arquivo com metadados) em uma pasta no sistema de arquivos. Quando a escrita do arquivo esperado termina, o mesmo é movido para uma pasta de trabalho e um novo *job* de entrada é criado no sistema de *workflow*.

O desenvolvimento deste componente foi realizado utilizando a linguagem Rust e é detalhado na Seção 6.2. Como o processo de escrita de arquivos grandes é, geralmente, demorado, foi necessária a criação de uma biblioteca que fosse capaz de identificar o fim da escrita do arquivo, os detalhes de implementação desta biblioteca se encontram na Seção 6.1.

Message Adapter

Para simplificar as interfaces dos *workers* e da API, estes componentes escutam, respectivamente, os protocolos AMQP e HTTP. Para que isto ocorra, é necessário que uma aplicação de adaptação exista pra traduzir as mensagens de *worklog* e controle de estado dos *jobs* para a API.

O Message Adapter escuta as filas de *worklog* e *workflow*, ligadas à *exchange* de resposta e envia as mensagens recebidas como *requests* HTTP para a respeitando a API REST do servidor web.

Este desenho habilita isolamento de infraestrutura entre *workers* e API, dificultando acessos indevidos aos *storages*, onde ficam as mídias.

Workers Entrada

É responsável pela entrada de mídias no MAM e possui 3 passos. Análise do arquivo de entrada, transcodificação dos vídeos de alta resolução e baixa resolução a serem inseridos no sistema, geração da imagem em miniatura do vídeo, conhecida como

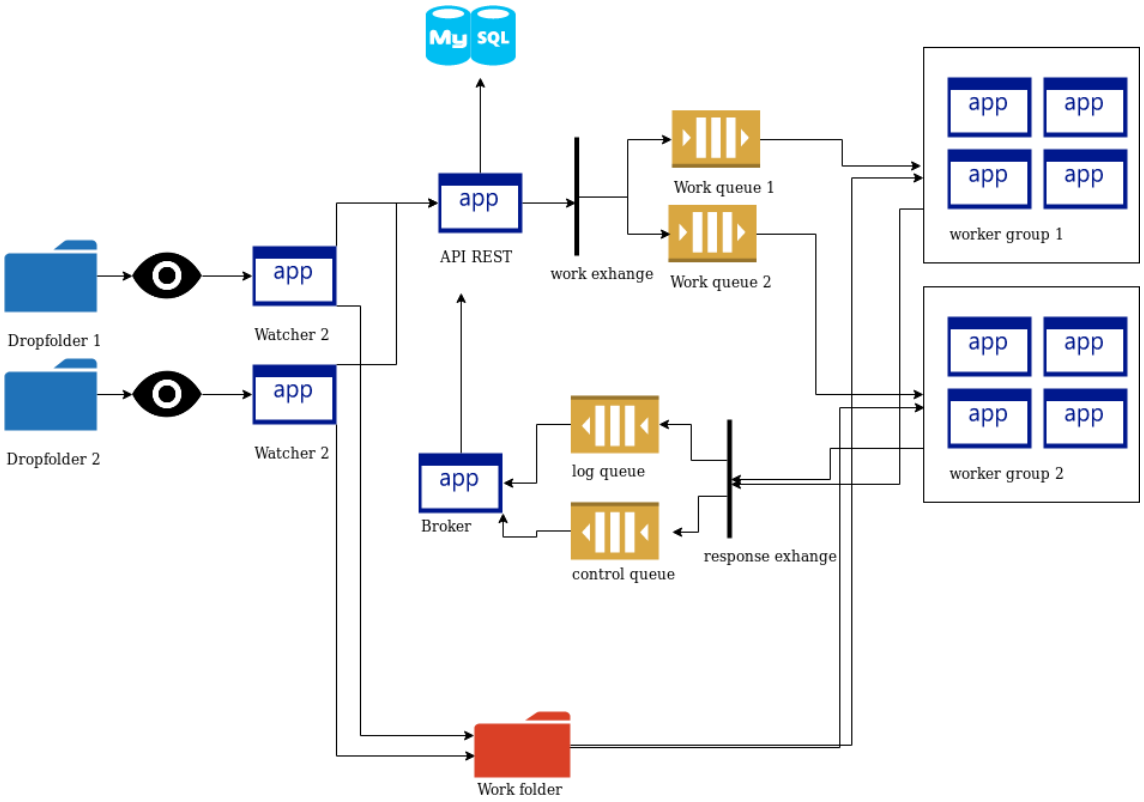

Figura 5.7: Arquitetura do sistema de *workflow*

thumbnail , transferência dos arquivos para o repositório de arquivos correspondente e entrada do ativo no MAM.

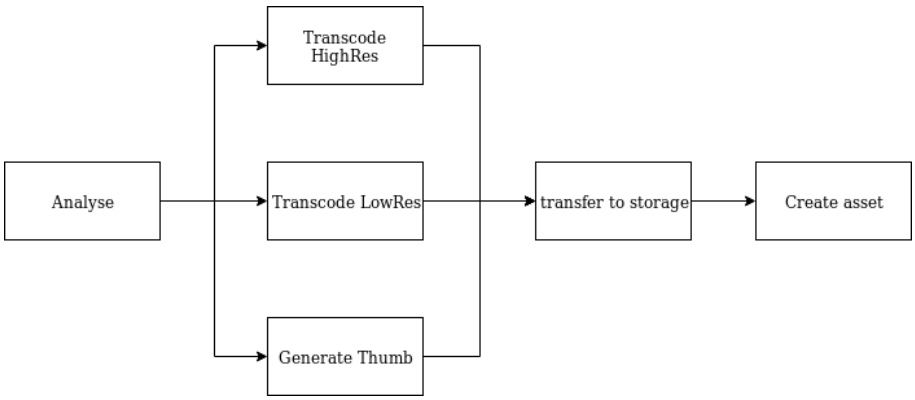

Figura 5.8: Fluxo de entrada

Worker de Pré-corte

É responsável pela entrega de cortes dos vídeos de alta resolução e disponibilização para o usuário e possui 4 passos. Transferência dos arquivos para o *storage* de trabalho, corte dos vídeos, envio para o repositório de *downloads* e entrada do recurso baixável no MAM.

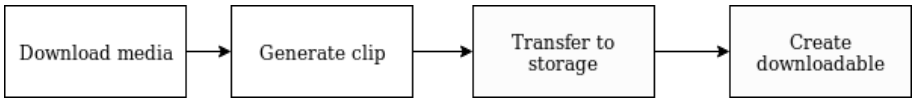

Figura 5.9: Fluxo de pré-corte

Worker Saída

Assim como o worker de Pré-corte, faz o corte de vídeos de alta resolução, porém em vez de expor um recurso baixável, envia o arquivo para um destino (*storage*) através de protocolo de transferência de arquivo.

Worker de quarentena

É responsável por arquivos originais relacionados à tarefas que falharam para um repositório de quarentena e possui 2 passos. Transferência do arquivo original (se houver) para o repositório de quarentena e entrada no MAM de recurso baixável.

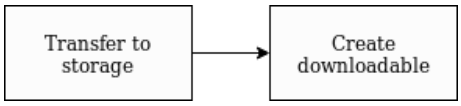

Figura 5.10: Fluxo de quarentena

Worker Index

Este worker faz a indexação de operações vindas da Assets API. Para que o processo seja mais performático no elasticsearch, é necessário o uso de inserções em lote (bulk) [22]. O Worker Index consome mais de uma mensagem da fila por vez para enviar as mensagens em bulk, se diferenciando dos demais que consome um *job* por vez.

5.3 Considerações Finais

Os componentes apresentados neste capítulo, MAM e *Workflow*, e sua arquitetura de microsserviços facilitam a implementação e implantação de um sistema altamente escalável, graças ao uso de microsserviços e do sistema de *workflow*. Ela também facilita a extensão pela adição de serviços ou customização pela substituição de serviços, e.g., para usar um motor diferente de busca, como o Solr, basta substituir (ou estender) a Search API e o Worker Index.

Capítulo 6

Decisões de implementação

O desenvolvimento de um sistema de gerenciamento de mídias envolve técnicas conhecidas e com vasto ecossistema, como aquelas voltadas para a web, porém envolve, também, problemas com menor número de ferramentas e bibliotecas existentes. Para resolver alguns destes problemas, foram criadas algumas soluções cujos detalhes de implementação são apresentados neste capítulo.

6.1 Stalker

Stalker é uma biblioteca escrita na linguagem Rust que tem por objetivo prover eventos de mudança no sistema de arquivos numa determinada pasta. Existem outras bibliotecas capazes de emitir eventos do sistema de arquivos, seja via sistema operacional, como o inotify no linux, seja por diferenças de estado dos arquivos na pasta. Estas bibliotecas, porém, não emitem eventos relacionados ao fim da escrita de um arquivo, ou estabilização. Por este motivo, foi desenvolvida a biblioteca stalker, pois em sistemas de fluxo de mídia são escritos grandes arquivos e os eventos de criação, escrita e remoção não são suficientes para mapear o fim do processo de transferência.

A biblioteca stalker utiliza a estratégia de comparação de estados do sistema de arquivos (*file system polling*). Esta abordagem foi escolhida pois, devido ao tamanho (grande) dos arquivos sendo transferidos, sistemas de fluxo de mídia utilizam montagens de sistemas de arquivos virtuais por rede, como nfs e cifs, além dos protocolos de transferência de arquivo (como FTP e SFTP), e no caso de sistemas de arquivo por rede, as chamadas de escrita, criação e deleção de arquivos não passam pelo *kernel* do sistema operacional, pois estão em outro sistema.

No modelo de *polling*, a pasta que está sendo observada tem seu estado guardado de tempos em tempos e comparados com o estado anterior. As mudanças são emitidas como eventos a serem utilizados, sendo eles: criação, modificação, movimentação, remoção e estabilização.

O evento de estabilização é emitido com base em uma métrica de estabilidade calculada no momento da extração do estado e um limite de tempo dado. A estabilidade é a diferença de tempo entre o momento da última modificação naquele arquivo e o momento em que o estado foi extraído. Se a estabilidade for maior que o limite definido, o arquivo é considerado estável, ou seja, sua escrita já foi concluída. O evento, porém, só é emitido se o arquivo não estiver estável na última comparação, evitando a reemissão do evento de estabilização.

6.2 Wozek

Para permitir a entrada de novas mídias no modelo assíncrono baseado em arquivos, foi criada a aplicação Wozek. Como citado na Seção 5.2.2, esta aplicação tem o papel de mover os arquivos de mídia escritos na pasta de entrada para a pasta de trabalho e iniciar o fluxo de entrada da mesma.

Esta aplicação foi desenvolvida na linguagem Rust, fazendo uso da biblioteca Stalker, anteriormente citada. Sua arquitetura possui 4 componentes principais:

- *Watcher*: Vindo da biblioteca stalker, este componente é responsável por observar a pasta de entrada e emitir eventos do sistema de arquivos.
- *Filter*: Filtra os eventos de fim de escrita e os padrões de nome de arquivos a serem processados.
- Parser: Processa um evento de sistema de arquivo e gera um comando de ingestão de arquivo.
- *Spawner*: Gerencia as *threads* de ingestão de arquivos (workers).
- *Log Engine*: Processa os eventos de *log* gerados nos outros componentes, serializa e persiste num repositório (arquivo ou Redis).
- *App*: Gerencia todo o ciclo de vida da aplicação e orquestra as ações na *thread* principal.

O ciclo de vida de um evento da aplicação começa pela sua criação no *Watcher* em sua *thread*. O evento é enviado para uma fila de sincronização e consumido pelo *Filter*, se o mesmo satisfizer as condições do filtro, é enviado para uma outra fila de sicronização que é consumida pelo Parser. No Parser, que possui sua própria *thread*, o evento é, então, processado e enviado um comando de ingestão para uma outra fila. Esta fila é consumida pelo Spawner, que possui uma *threadpool* de *workers*. O Spawner, então envia o comando para a *threadpool* e o mesmo é executado por um dos *workers* presentes.

Esta implementação faz forte uso do módulo "sync::mcsp::channel" da biblioteca padrão da linguagem Rust, que expõe duas estruturas de escuta e escrita que facilitam a comunicação assíncrona entre *threads*.

Todos os componentes da aplicação possuem acesso a uma *facade* de *log* que envia eventos de log para a *Log Engine*. É possível então rastrear o funcionamento e o estado da aplicação.

Para a implementação das lógicas de Parse e Filter, foram implementadas duas formas de configuração, fixa e dinâmica. A configuração fixa, no caso do Filter significa a definição de padrões na forma de 'globs' e no caso do Parse, expressões regulares de substituição ou valores fixos para cada campo do comando de ingestão. A configuração dinâmica utiliza a linguagem de script embarcada Gluon tanto para o Filter, através da implementação de uma função que recebe o evento e retorna verdadeiro ou falso, quanto para o Parser através da implementação de uma função que retorna o comando de ingestão.

6.3 Workflower

Workflower é um sistema/*framework* para desenvolvimento de *workflows* assíncronos simples criado inicialmente para o uMAM, embora possa ser usado para outros propósitos. Ele foi desenvolvido na linguagem python e possui dois componentes principais de controle, API e *Message Adapter*. Ele utiliza o banco de dados MySQL para persistir as informações dos *jobs* e o *broker* RabbitMQ para mensagens assíncronas.

O componente API implementa todas as regras de controle de estado e criação de *jobs*. Para tal, ele expõem uma API REST que expõe os seguintes recursos:

- *Job*: Um processamento enviado a ser realizado.
- *Worklog*: Uma mensagem de *log* enviada pelo *worker* para acompanhamento do processamento do *Job*
- Status: É uma enumeração de status possíveis para um *Job*
- *Workflow*: É uma definição de entrada, onde são definidos uma fila inicial e dados padrão do *Job*

Para facilitar o isolamento entre as entradas de *jobs* e os workers, não existe comunicação direta entre entre *workers* e API. Toda mensagem de resposta (*worklog*, atualização de status, atualização de step e redirecionamento) são enviadas para filas de resposta, que são, então, consumidas pelo *Message Adapter* e enviadas para a API via HTTP. Foi desenvolvida a biblioteca bureaucrat descrita abaixo para facilitar o desenvolvimento.

6.4 Bureaucrat

A biblioteca Bureaucrat foi desenvolvida para facilitar e normalizar o desenvolvimento de *workers* no Workflower. Seu *framework* define 3 conceitos:

- *Worker*: É a classe principal onde estão implementadas as regras do ciclo de vida da aplicação
- Step: Define um passo do *workflow*
- JobHandler: É a representação do *Job*. Ele contém seus dados e sua representação para o resto do sistema

O uso da biblioteca se dá pela construção de um objeto da classe *Worker* e definição de classes chamadas *Step*. Os *steps* são então passados em ordem para o *worker*.

Um objeto *Step* representa uma etapa no workflow e deve possuir um método chamado *run* recebendo um objeto do tipo *JobHandler* como argumento, como pode ser visto no trecho abaixo. Quando um *job* chega para ser processado, o *worker* cria um objeto JobHandler que é passado para cada Step pelo método *run*. O objeto *job_handler* possui métodos para a comunicação com o Workflower como worklogs, atualização de status, além de possuir os dados da mensagem do job.

```
from bureaucrat import Worker, JobHandler
from bureaucrat.effects import Redirect

CALCULATING_SUM_STATUS_ID = 2
CALCULATING_FACTORS_STATUS_ID = 3

class SumStep:
    def run(self, job_handler: JobHandler):
        job_handler.worklog.info('Starting sum process')
        job_handler.update_status(CALCULATING_SUM_STATUS_ID)
        try:
            result = sum(job_handler.data['numbers'])
            job_handler.work_data['sum'] = result
        except Exception as exc:
            job_handler.worklog.error('Sum has gone wrong')
            raise Redirect('dead-letter')

class FactorsStep:
    def run(self, job_handler: JobHandler):
        job_handler.worklog.info('Starting factors process')
        job_handler.update_status(CALCULATING_FACTORS_STATUS_ID)
        try:
            input = job_handler.data['sum']
            result = filter(range(input), lambda i: not input % i)
            job_handler.work_data['sum'] = result
        except Exception as exc:
            job_handler.worklog.error('Sum has gone wrong')
            raise Redirect('dead-letter')

worker = Worker(...)
worker.steps = [
    SumStep(),
    FactorsStep(),
]
```

A divisão do workflow em *Steps* permite a reutilização dos mesmos para diferentes *workflows*, agilizando ainda mais o desenvolvimento de novos *workers*.

6.5 Considerações Finais

Nesse capítulo, vimos as tecnologias desenvolvidas nesse trabalho que serviram como habilitadoras das diversas partes do sistema. Embora as mesmas, como no caso do Wozek ou do Workflower, possuam complexidade superior à do próprio sistema, o seu desenvolvimento independente permite que estas tecnologias sejam usadas para outros fins, trazendo um benefício colateral ao projeto.

Capítulo 7

Modelos de operação

Para que um sistema aberto seja utilizado, é necessário que ele seja implantado, operado e administrado. Devido às complexidades inerentes a um sistema de mídia, com múltiplos apontamentos de storage (cada worker deve possuir um ponto de montagem para cada storage e cada storage deve permitir a montagem) e serviços, a simples distribuição e instalação do sistema não é suficiente para permitir o uso do sistema.

Nesse capítulo, são abordados modelos e técnicas de implantação e operação de sistemas a fim de diminuir a fricção causada pela complexidade do sistema e permitir o seu amplo uso.

7.1 Devops

A fim de diminuir os ciclos de entrega e aumentar a resiliência de sistemas, algumas organizações adotam uma cultura conhecida como Devops [23]. A expressão vem da junção das palavras development e operations (desenvolvimento e operações) e define uma cultura voltada para o aumento da empatia e diminuição da distância entre times de desenvolvimento e operações.

A cultura Devops traz práticas dos dois mundos para atingir seus objetivos. Ela traz práticas de automação, codificação e versionamento de definições de infraestrutura vindas do desenvolvimento de software e a preocupação com a resiliência do sistema para o processo de desenvolvimento. Do direcionamento para práticas de desenvolvimento, surgiram conceitos de infraestrutura como código e foram desenvolvidas ferramentas para apoiar estes processos.

Os pilares/áreas do Devops são conhecidos como CALMS, um acrônimo para Culture, Automation, Lean, Monitoring e Sharing [24]. Culture representa a adoção por parte da organização da cultura Devops. Automation está ligado à adoção de ferramentas que contribuam com a diminuição de tarefas manuais na operação dos sistemas, bem como ao gerenciamento da infraestrutura de maneira semelhante à forma como se gerencia código (infra as code). Lean se refere a gerência do trabalho, minimizando o trabalho a ser feito para executar tarefas e diminuindo os ciclos de entrega. Monitoring, por sua vez, é a prática de técnicas de monitoração a fim de aumentar a identificação ativa de problemas e incidentes, diminuindo, assim, a identificação passiva e soluções reativas emergenciais. Sharing se refere ao foco em comunicação que tem por objetivo diminuir a distância entre times de operação e desenvolvimento.

Nesse trabalho, sempre que possível, foram utilizadas técnicas relacionadas a este conceito a fim de diminuir a carga operacional inerente a um MAM.

7.2 Implantação

Para realizar a implantação do sistema e provisionamento do sistema operacional, foi utilizada a ferramenta Ansible (apresentada na Seção 3.5). Foram criados *playbooks* para fazer a instalção dos serviços desenvolvidos, bem como aplicações usadas como recursos, como o banco de dados, servidor de HTTP e mensageria. Os plabooks criados também permitem o gerenciamento de apontamento e acessos de storage, indispensável para o funcionamento do sistema.

Além destas configurações, é necessário o gerenciamento dos servidores e recursos, físicos ou virtuais. Este gerenciamento varia de acordo com o tipo de infraestrutura usada. Nas seções a seguir, são abordadas as características do gerenciamento de estruturas própria (física) ou em nuvem (pública).

7.3 Infraestrutura Própria

7.3.1 Armazenamento de arquivos

Para garantir a operação e integridade do sistema, é necessário que se mantenha a disponibilidade das mídias nele presentes. Para tal, devem ser empregadas técnicas e estratégias de redundância, e.g., duplicação de conteúdo. Caso seja viável, a duplicação de conteúdo deve ocorrer em infraestruturas distintas para prevenção de desastre. Outra estratégia para o armazenamento de arquivos que simplifica a operação do sistema é o uso de um sistema de storage distribuído como o CEPH. Ele garante os princípios de replicação e disponibilidade de forma transparente, além de desacoplar a operação relacionada a armazenamento da operação do sistema [25].

7.4 Nuvem

A computação em nuvem é um conceito abrangente que designa a abstração de componentes de infraestrutura de hardware, rede, software e até plataformas tecnológicas, permitindo seu provisionamento remoto e automático (orquestrado por software) [26]. Soluções em nuvem são comumente utilizadas afim de facilitar o gerenciamento dos recursos computacionais e diminuir o custo inicial de implantação de um sistema, dado que não é necessária a aquisição de equipamentos e manutenção de infraestrutura. Nesta seção serão abordadas algumas características da implantação do sistema em uma nuvem pública: a Amazon Web Service (AWS).

7.4.1 Infraestrutura de base

Como estão sendo usados recursos de infraestrutura como serviço (IaaS), é necessário especificar o padrão de arquitetura comum em todos os serviços (que respondem HTTP) implantados. O objetivo de definir tal arquitetura base é permitir que todos os serviços possuam escalabilidade e tolerância a falhas. A infraestrutura base usa os seguintes serviços de infraestrutura:

- EC2: Serviço de máquinas virtuais da AWS
- Auto Scaling Group (ASG): Serviço que provisiona automaticamente instancias de EC2 definidas, baseado em configurações de gatilho
- Elastic Loadbalancer (ELB): Serviço de balanceamento de carga que, integrado com o ASG, permite a comunicação dos consumidores dos serviços com as instâncias dos mesmos

Cada serviço tem então a instalação de sua aplicação replicada em suas instâncias. Estas, por sua vez, são controladas pelo ASG que usa gatilhos de carga vindos do ELB para aumentar ou diminuir o número de instâncias ativas. O ELB serve como único canal de comunicação com as aplicações daquele serviço.

São configurados, também, serviços de rede virtual, como subnets e security groups (parecidos com regras de firewall), a fim de permitir a comunicação e segurança do sistema, quando necessário.

7.4.2 Serviços

Para implantar os serviços mencionados no capítulo 5, como SGBD, cache em memória e API Gateway, foram utilizados serviços da AWS que correspondem a cada um destes elementos, sendo eles:

- Relational Database Service (RDS): Oferece instâncias gerenciadas contendo vários SGBD, incluindo o MySQL utilizado neste projeto
- ElastiCache: Oferece cache em memória como serviço, obedecendo o protocolo exposto pelos softwares Redis e Memcached
- API Gateway: Serviço de gerenciamentos de APIs, oferecendo abstração, segurança e monitoramento das mesmas.

7.5 Considerações Finais

Vimos neste capítulo as diferentes possibilidades na implantação do sistema. Apoiadas pelos conceitos do CALMS, ambas podem ter seus custos iniciais e de crescimento diminuídas. Devido à característica de oferecer seus recursos como serviço, a implantação em nuvem permite gerenciamento mais fácil dos recursos quando se utiliza conceitos como infra como código, pois existem ferramentas que apoiam este processo.

Vale lembrar que em termos de custos a solução em nuvem tem um custo inicial menor com itens físicos já que não há necessidade de gerenciamento de hardware (manutenção, ciclo de obsolescência, etc), além de diminuir o conhecimento necessário dentro da organização. Os custos na nuvem também aumentam conforme o uso, fato que exige acompanhamento constante.

Em pequenas instalações, o uso de infraestrutura própria pode ser indicado pois, apesar de possuir um custo inicial mais alto, possui custos fixos mais baixos e necessita de menor controle financeiro.

Capítulo 8

Conclusão

As mudanças ocorridas nas formas de consumo de mídias audiovisuais tiveram a tecnologia como agente de transformação, seja na distribuição (utilizando plataformas disponíveis na internet), seja no consumo em si (por meio de dispositivos móveis, por exemplo). Estas mudanças associadas à facilidade de publicação em plataformas digitais ampliou o mercado de mídia para novos produtores de conteúdo, para além daqueles que produzem para grandes meios, como a TV.

Para permitir que os produtores independentes (indivíduos ou pequenas organizações) obtenham agilidade e gerenciamento na produção, assim como é possível na TV, foi proposto o desenvolvimento de um sistema de MAM. Foram, então levantadas as características dos MAMs existentes. Para o sistema proposto, foram definidas funcionalidades que fossem compatíveis com aquelas dos sistemas existentes, mas que atendessem o pequeno produtor (tendo premissas de acessibilidade e baixo custo).

Visando implementar o sistema descrito, foi feito o desenho da arquitetura do sistema e descrição dos seus serviços. Foram implementados, ainda, alguns componentes de mais baixo nível a fim de habilitar algumas funcionalidades do sistema cujas características de implementação são descritas no Capítulo 6.

Dada a complexidade de operação do sistema, foi necessário abordar as decisões de operação no Capítulo 7. Esta elucidação foi necessária para tentar garantir que a complexidade do sistema tenha um impacto menor nas premissas de acessibilidade e baixo custo.

8.1 Contribuições

Como identificado nos capítulos anteriores, não havia um sistema que atendesse as necessidades de gerenciamento de mídia para a produção do pequeno produtor de conteúdo. O sistema desenvolvido trouxe contribuições que são descritas a seguir.

8.1.1 Formalização Acadêmica

Os levantamentos realizado sobre as funcionalidades de um MAM, são escassos e, geralmente, antigos. O levantamento realizado neste trabalho pode contribuir para outros trabalhos ou pesquisas relacionadas a este tipo de sistema.

8.1.2 Open source

O sistema desenvolvido possui código aberto e usa tecnologias de código aberto. Isto possibilita não só o uso gratuito da solução, como possibilita a sua extensão e modificação de suas funcionalidades e funcionamento. Da mesma forma, as tecnologias desenvolvidas possuem licença de código aberto, possibilitando seu uso para outros fins.

8.2 Limitações

Apesar de alcançar contribuições importantes, o projeto possui limitações tanto internas (característica da execução do projeto), quanto externas (inerentes ao assunto abordado). Estas limitações são abordadas a seguir.

8.2.1 Referências de Modelo Atual

Nas pesquisas relacionadas aos MAMs existentes não foi encontrada nenhuma fonte formal de informação, sendo necessário levantar as características nas páginas dos produtos na web.

8.2.2 Usabilidade

A usabilidade, seja sob o ponto de vista estético, seja sob o ponto de vista ergonômico, não foi abordada no desenvolvimento do sistema e nem no levantamento dos sistemas existentes. Esta limitação fere a premissa de agilidade da solução e limita o alcance aos usuários.

8.2.3 Enriquecimento de conteúdo

O sistema proposto carece de um subsistema de enriquecimento, manual ou automático, de metadados dos seus *assets*. O enriquecimento de conteúdo, como é o caso do *logging* descrito no Capítulo 2, é essencial para tornar as buscas mais acuradas para os casos de uso específicos do sistema.

8.2.4 Complexidade Operacional

Uma das premissas do sistema era a de acessibilidade e, como descrito no Capítulo 7, a própria natureza do sistema torna sua operação complexa. Embora as técnicas descritas diminuam a fricção, a operação do sistema ainda necessita de um corpo técnico especializado.

8.3 Trabalhos Futuros

Embora o sistema desenvolvido atenda o uso mínimo para pequenos ambientes de produção, algumas funcionalidades e características permitiriam o uso mais efetivo e atenderiam necessidades comuns a estes usuários. Nas sessões a seguir, são descritas as funcionalidades e pesquisas que contribuiriam para o aperfeiçoamento do sistema.

8.3.1 Controle de direitos autorais

Na produção audiovisual, grande ou pequena, nem todo material usado é original. Sobre estes materiais de terceiros, recaem regras, sejam elas de divulgação de autoria, pagamento de *royalties*, etc.

Para os conteúdos distribuídos em plataformas digitais existem, inclusive, leis internacionais, como a Lei europeia de *copyright*. Esta lei diz em seu artigo 13 que as plataformas devem oferecer às empresas detentoras de direitos autorais meios automáticos de detecção de conteúdo [27]. Isto permite que estas empresas possam reclamar direitos sobre conteúdos, mesmo que o conteúdo não original seja apenas uma parte do todo.

Para mitigar problemas relacionados à autoria, o MAM deveria dispor de meios, automáticos ou manuais de gerenciamento de metadados de direitos autorais para os *assets* gerenciados.

8.3.2 Modelo SaaS

Para vencer o desafio de tornar o sistema acessível para seus usuários, uma das estratégias possíveis é adotar um modelo que centralize as complexidades operacionais e as abstraia dos usuários. O modelo SaaS (*Software as a Service*) permite que uma única entidade/organização opere o sistema, enquanto outras o utilizam.

Para que o sistema possa ser usado no modelo SaaS (comercialmente ou de maneira coletiva), é necessário que o sistema sofra adaptações que possibilitem as seguintes características:

- distribuição de cotas de armazenamento
- distribuição de cotas de uso de processamento computacional (*ingest*, corte, enriquecimento de metadados, etc)
- Grupos de usuários
- Permissões de acesso baseadas em Grupos de Acesso (ACL)

8.3.3 Solução móvel para *ingest*

Em alguns cenários de uso do sistema, a possibilidade de compartilhamento remoto de conteúdo se faz necessária para garantir a agilidade do processo de produção. Sendo assim, uma solução na forma de aplicativo móvel permitiria o envio de materiais capturados diretamente para o sistema, sem depender de outros atores intermediários e de forma integrada.

8.3.4 Mecanismo de enriquecimento de dados

Para permitir buscas mais acuradas do material armazenado, o sistema deve ser capaz de prover mecanismos de enriquecimento de metadados dos materiais. Algumas formas possíveis de enriquecimento são:

- Transcrição automática de conteúdo
- Identificação de personagens

- Identificação de texto no vídeo
- Identificação de objetos

Estes e outros tipos de enriquecimentos inteligentes, associados a conexão com bases de conhecimento podem modificar profundamente a produção de conteúdo. A fim de habilitar tais funcionalidades, o sistema poderia prover mecanismos capazes de permitir o enriquecimento, por ferramentas internas ou de terceiros, dos metadados do material armazenado.

8.3.5 Mapeamento de processos

As características escolhidas para o sistema foram levantadas pela observação externa da produção de conteúdo para as plataformas de distribuição. A descoberta das reais necessidades dos usuários do sistema poderia ser atingida pelo mapeamento dos processos de produção específicos destes produtores, seja pela observação do seu trabalho, seja através de entrevistas.

8.3.6 Avaliação do sistema

Para validar se o sistema entrega aos seus usuários funcionalidades que realmente garantem qualidade e agilidade no seu processo de produção de conteúdo, seria necessário avaliar o uso do mesmo. Para tal, podem ser usadas técnicas como coleção de métricas de uso (por instrumentação do sistema) ou entrevistas com os usuários.

Referências Bibliográficas

[1] AVID. "Media Central Screen Shot". . Disponível em: <https://www.avid.com/-/media/avid/images/products/mediacentral/mediacentralcollage_1200x607.jpg?h=607&la=en&mw=1200&w=1200&v=20180802174335&hash=02D9C578C18126FED2730E2487AFAB954592B4C4&h=607&la=en&mw=1200&w=1200&v=20180802174335&hash=02D9C578C18126FED2730E2487AFAB954592B4C4>.

[2] AVID. "Avid - Media Central Features". . Disponível em: <https://www.avid.com/products/mediacentral/features>.

[3] BEAT, B. "Nexidia IBC2015 Preview". Disponível em: <https://www.broadcastbeat.com/nexidia-ibc2015-preview/>.

[4] VIZRT. "Viz One Datasheet". Disponível em: <https://dam.vizrt.com/Dmm3BWSV3/assetstream.aspx?assetid=2797&AssetOutputIdent=44&accesskey=9e9455c5-270d-47de-8bcc-863f7c1177b0&download=true>.

[5] BARNOUW, E. *Tube of Plenty: The Evolution of American Television.* 2 ed. New York, Springer-Verlag, 1990.

[6] BENIGER, J. *The Control Revolution: Technological and Economic Origins of the Information Society.* 2 ed. New York, Springer-Verlag, 2009.

[7] AUSTERBERRY, D. *Introduction to Digital Asset Management.* 2 ed. New York, Springer-Verlag, 1988.

[8] COPPE. "COPPE TV". Disponível em: <https://www.coppetv.coppe.ufrj.br>.

[9] AWS. "AWS prices definition for S3". Disponível em: <https://aws.amazon.com/s3/pricing/>.

[10] AVID. "Avid - iNEWS". . Disponível em: <https://www.avid.com/products/inews>.

[11] AVID. “Avid - Media Central - Third Party Connectors”. . Disponível em: <https://www.avid.com/products/mediacentral/build-your-own#Third-party-connectors>.

[12] MANTON, J. *Avalon Media System (review).* Relatório técnico, 2017.

[13] SMPTE. “About SMPTE”. Disponível em: <https://www.smpte.org/about>.

[14] DELVIN, B. *What is MXF?* Relatório técnico, EBU Technical Review, 2002.

[15] VAN ROSSUM, G., OTHERS. “Python Programming Language.” In: *USENIX Annual Technical Conference*, v. 41, p. 36, 2007.

[16] TEAM, P. “Flask Documentation”. Disponível em: <http://flask.pocoo.org/docs/1.0/>.

[17] HEAP, M. “Advanced Ansible”. In: *Ansible*, Springer, pp. 137–157, 2016.

[18] DEVELOPERS, F. “About FFmpeg”. . Disponível em: <https://www.ffmpeg.org/about.html>.

[19] DEVELOPERS, F. “About FFmpeg”. . Disponível em: <https://www.ffmpeg.org/about.html>.

[20] NUXT.JS. “Nuxt.JS - Universal Vue.JS Applications”. Disponível em: <https://nuxtjs.org/>.

[21] BREHEM, S. “Isomorphic Javascript: The future of web apps”. 2013.

[22] ELASTIC.CO. “Tunning for Indexing Speed | Elasticsearch reference”. Disponível em: <https://www.elastic.co/guide/en/elasticsearch/reference/master/tune-for-indexing-speed.html#_use_bulk_requests>.

[23] DYCK, A., PENNERS, R., LICHTER, H. “Towards definitions for release engineering and devops”. In: *Release Engineering (RELENG), 2015 IEEE/ACM 3rd International Workshop on*, pp. 3–3. IEEE, 2015.

[24] HAMUNEN, J. *Challenges in adopting a Devops approach to software development and operations.* G2 pro gradu, diplomityö, 2016. Disponível em: <http://urn.fi/URN:NBN:fi:aalto-201609083476>.

[25] WEIL, S. A., BRANDT, S. A., MILLER, E. L., etal. “Ceph: A scalable, high-performance distributed file system”. In: *Proceedings of the 7th symposium*

on Operating systems design and implementation, pp. 307–320. USENIX Association, 2006.

[26] VAQUERO, L. M., RODERO-MERINO, L., CACERES, J., etal. "A break in the clouds: towards a cloud definition", *ACM SIGCOMM Computer Communication Review*, v. 39, n. 1, pp. 50–55, 2008.

[27] COMMISSION, E. "DIRECTIVE OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL on copyright in the Digital Single Market". Disponível em: <https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:52016PC0593&from=EN>.