Instituto Politécnico de Tomar

Instituto Politécnico de Tomar

MERmaid v0.3 – Deep Flow

Projeto

Luís Henrique Soares de Albergaria Almeida Costa

Mestrado Engenharia Informática

Internet das Coisas

Tomar. Novembro, 2025

Esta página foi intencionalmente deixada em branco.

Instituto Politécnico de Tomar

Instituto Politécnico de Tomar

MERmaid v0.3 – Deep Flow

Projeto

Luís Henrique Soares de Albergaria Almeida Costa

Orientado por:
Professor Renato Panda, Instituto Politécnico de Tomar

Projeto apresentado ao Instituto Politécnico de Tomar para cumprimento dos requisitos necessários à obtenção do grau de Mestre (Mestrado Engenharia Informática)

Esta página foi intencionalmente deixada em branco.

Informação sobre o Autor, Orientador e Financiamento

Autor:	Luís Henrique Soares de Albergaria Almeida Costa aluno25647@ipt.pt
Orientadores:	Prof. Renato Eduardo Silva Panda renato.panda@ipt.pt
Instituição de acolhimento:	Ci2 - Smart Cities Research Center
Instituição que concede o grau:	Instituto Politécnico de Tomar

Financiamento:	MERGE - PTDC/CCI-COM/3171/2021 Ci2 Base Funding - UIDB/05567/2020

This work is funded by FCT - Foundation for Science and Technology, I.P., within the scope of the projects: MERGE (Music Emotion Recognition - Next Generation), DOI: 10.54499/PTDC/CCI-COM/3171/2021 financed with national funds (PIDDAC) via the Portuguese State Budget; and Ci2 Base and Programmatic Funding - UIDB/05567/2020 and UIDP/05567/2020, with funds from the European Social Fund, through the Regional Operational Program Centro 2020.

Esta página foi intencionalmente deixada em branco.

Dedico este trabalho a todos que me apoiaram
ao longo deste percurso académico.

Learn from the ones who came before,
and lay the trail, for the ones who come after.
— Gustave

Esta página foi intencionalmente deixada em branco.

Agradecimentos

Em primeiro lugar, agradeço à minha família, que me apoiou ao longo de todo o mestrado e, em particular, ao meu avô, que contribuiu para a revisão da escrita deste relatório.

Em segundo lugar, agradeço ao professor Renato Panda por ter aceite orientar-me no desenvolvimento do projeto.

Expresso também o meu reconhecimento ao Instituto Politécnico de Tomar e aos seus docentes, por proporcionarem uma experiência académica enriquecedora e um ambiente favorável à aprendizagem e à investigação.

Por fim, deixo uma palavra de agradecimento aos meus colegas Paulo Peixoto e Ricardo Elisiário por, mesmo não tendo nós trabalhado sempre juntos, mantivemos um ambiente de apoio coletivo, espírito de grupo e camaradagem.

Esta página foi intencionalmente deixada em branco.

Abstract

This work presents the development of `MERmaid v0.3`, a modular audio-processing platform designed for music research and multimodal analysis. The system was restructured according to a microservices architecture, enabling each stage of the processing pipeline — download, segmentation, source separation, automatic transcription, genre classification, emotion classification and waveform generation — to operate independently, with improved scalability and replaceability. Asynchronous communication between services is ensured by RabbitMQ, while global state management and result persistence are handled by a dedicated *Pipeline Manager* backed by MongoDB.

The development of this version focused on the complete restructuring of core services, the stabilisation of the processing flow, and the unification of communication interfaces. Both the API and the Web application were adapted to support the new orchestration model, ensuring stronger security guarantees and consistent interaction between the front-end and the distributed processing backend.

The resulting platform provides a reliable foundation for academic work, enabling experimentation with different *Machine Learning* models, comparative analysis between transcriptions, exploration of alternative preprocessing strategies and the integration of future analytical modules. Although the current version prioritises robustness over advanced functionality, the system was designed with extensibility in mind, including support for new audio formats, alternative separation and transcription models, automatic metric computation and deeper integration with musical analysis tools. Altogether, these contributions establish a solid basis for future research and for the continued evolution of the platform across subsequent versions.

Keywords: audio processing, microservices, automatic transcription, source separation, music analysis, Whisper, Demucs, Librosa, RabbitMQ.

Esta página foi intencionalmente deixada em branco.

Resumo

Este trabalho apresenta o desenvolvimento da versão `MERmaid v0.3`, uma plataforma modular de processamento de áudio orientada para investigação musical e análise multimodal. O sistema foi reestruturado segundo uma arquitetura de microsserviços, permitindo que cada etapa do *pipeline* — *download*, segmentação, separação de fontes, transcrição automática, classificação de género, classificação emocional e geração de forma de onda — funcione de forma independente, escalável e substituível. A comunicação assíncrona entre serviços é assegurada pelo RabbitMQ, enquanto a gestão de estado global e a persistência de resultados são delegadas a um *Pipeline Manager* com armazenamento em MongoDB.

O desenvolvimento desta versão focou-se na reimplementação completa de serviços centrais, na estabilização do fluxo de processamento e na unificação das interfaces de comunicação. A API e a aplicação Web foram igualmente adaptadas, passando a suportar o novo modelo de orquestração, garantindo maior segurança e consistência entre o front-end e o processamento distribuído.

A plataforma resultante constitui uma base fiável para trabalho académico, permitindo a experimentação com diferentes modelos de *Machine Learning*, análise comparativa entre transcrições, exploração de métodos de pré-processamento e futuras extensões analíticas. Apesar da versão atual privilegiar a estabilidade sobre funcionalidades avançadas, o sistema foi pensado para evoluir, incluindo suporte para novos formatos de áudio, modelos alternativos de separação e transcrição, cálculo automático de métricas e integração mais profunda com ferramentas de análise musical. Em conjunto, estes contributos transformam a plataforma numa fundação sólida para a investigação futura e para o desenvolvimento das próximas versões do sistema.

Palavras-chave: processamento de áudio, microsserviços, transcrição automática, separação de fontes, análise musical, Whisper, Demucs, Librosa, RabbitMQ.

Índice

Esta página foi intencionalmente deixada em branco.

Lista de Figuras

Esta página foi intencionalmente deixada em branco.

Lista de Tabelas

Esta página foi intencionalmente deixada em branco.

Lista de Excertos de Código

Esta página foi intencionalmente deixada em branco.

Lista de Abreviaturas

Acrónimo	Forma extensa	
API	Application Programming Interface (Interface de Programação de Aplicações)	(p. 2), (p. 10), (p. 12), (p. 16), (p. 24), (p. 27), (p. 29), (p. 31–33), (p. 35), (p. 38), (p. 43), (p. 50), (p. 56–58), (p. 61, 62)
CNN	Convolutional Neural Networks (Redes Neuronais Convolucionais)	(p. 8, 9)
DL	Deep Learning (Aprendizagem Profunda)	(p. 8), (p. 10), (p. 14), (p. 29)
k-NN	k-Nearest Neighbors (k-Vizinhos Mais Próximos)	(p. 8, 9)
MER	Music Emotion Recognition (Reconhecimento de Emoções em Música)	(p. 1), (p. 5, 6), (p. 8), (p. 12–14), (p. 23), (p. 28, 29)

ML	Machine Learning (Aprendizagem Automática)	(p. 8, 9), (p. 14), (p. 32)
OCR	Optical Character Recognition (Reconhecimento Ótico de Caracteres)	(p. 12)
RNN	Recurrent Neural Networks (Redes Neuronais Recorrentes)	(p. 8, 9)
SSH	Secure Shell	(p. 21)
SVM	Support Vector Machines (Máquinas de Vetores de Suporte)	(p. 8, 9)
VS Code	Visual Studio Code	(p. 19), (p. 21)
WER	Word Error Rate (Taxa de Erro de Palavras)	(p. 45), (p. 47), (p. 62)
WSL2	Windows Subsystem for Linux 2	(p. 19, 20), (p. 22)

Capítulo 1

Introdução

Este capítulo apresenta o contexto, a motivação e os objetivos do *MERmaid v0.3*, ao referir contributos passados e presentes, descrevendo sucintamente o problema.

1.1 Contexto e Motivação

A música faz parte do dia a dia de variados povos e culturas, e a internet é cada vez mais utilizada para fácil acesso a este *media*. Aplicações ou serviços como Spotify[1],YouTube[2] e Apple Music[3], compostas por extensas bases de dados musicais, revelam uma crescente adesão aos seus serviços, devido à sua interface intuitiva e conveniência (Guo, 2023).

A musicalidade é inerente ao ser humano, faz parte dos ritmos naturais do corpo e da emoção, servindo como meio primordial de ligação e partilha, e é através dela que construímos empatia, memória e identidade (Malloch & Trevarthen, 2018). Surge, então, como campo de investigação que procura explorar a ligação entre a arte e a tecnologia, o Music Emotion Recognition (Reconhecimento de Emoções em Música) (MER), de forma a classificar e categorizar as diversas emoções transmitidas pela música, tornando possível melhorar o uso pessoal, comercial e até mesmo clínico.

Esta é uma área de estudo em crescimento e apresenta elevado potencial, mas enfrenta ainda desafios significativos que limitam o seu avanço. Um dos principais obstáculos é a disponibilidade reduzida de *datasets* amplos, diversificados e bem anotados, essenciais para treinar modelos de reconhecimento de emoções, garantindo que conseguem lidar com a vasta diversidade de géneros musicais, culturas e interpretações emocionais. Esta limitação afeta diretamente a capacidade de os sistemas serem abrangentes e eficazes.

Outro obstáculo ao estudo da área MER reside na dificuldade de capturar e

[1] `https://www.spotify.com`
[2] `https://www.youtube.com`
[3] `https://music.apple.com`

traduzir as características musicais de forma eficaz. A música apresenta variados elementos, como ritmo, harmonia, melodia e dinâmica, capazes de provocar emoções. Representar esta complexidade através de algoritmos computacionais exige métodos inovadores e a articulação entre áreas como música, psicologia e processamento de dados.

Para além destes desafios, destaca-se ainda a escassez de sistemas concretos que permitam validar estas abordagens na prática. Ainda que promissor, o desenvolvimento de protótipos funcionais que integram o reconhecimento de emoções em música ainda é reduzido, sendo que a ausência destas ferramentas prejudica a validação prática das metodologias desenvolvidas e o seu progresso científico.

Neste contexto, surge o *MERmaid v0.3*, uma continuidade do projeto MERmaid em desenvolvimento pelo Music Information Retrieval Research Group, no Centro de Informática e Sistemas da Universidade de Coimbra, em colaboração com o Smart Cities Research Center/DataLab, do Instituto Politécnico de Tomar. Este sistema funcional, disponibilizado a um grupo fechado de investigadores, procura criar um sistema que permita demonstrar as possibilidades da área em estudo e facilite o avanço da mesma através de um ambiente de teste de resultados.

1.2 Objetivos e Contribuições

A iteração atual do projeto procurou continuar o desenvolvimento do projeto MERmaid e tornar funcional o protótipo atual. Acima de tudo, o trabalho permitiu garantir que o sistema atingisse um nível de funcionalidade que possibilitasse uma utilização prática para além da camada de apresentação Web, oferecendo funcionalidades de processamento de sinal para um grupo restrito de investigadores.

Embora os objetivos definidos incluíssem a melhoria da estabilidade geral e a expansão das capacidades analíticas, o trabalho realizado ultrapassou essa meta inicial, resultando na reconstrução estruturada de grande parte do sistema e na introdução de um *pipeline* de processamento totalmente operacional.

As principais contribuições deste trabalho incluem a implementação integral do serviço *Manager*, responsável pela orquestração de todas as etapas do *pipeline* e pela persistência do estado de cada processamento, bem como o desenvolvimento ou reimplementação de um conjunto de microsserviços containerizados: *download* robusto via YouTube-DLP; segmentação de áudio; separação de fontes com Demucs; transcrição com Whisper; geração de forma de onda; classificação emocional e classificação de género. Cada serviço foi isolado num ambiente próprio, permitindo a substituição ou evolução independente dos restantes, garantindo assim a escalabilidade e a extensibilidade do sistema.

Paralelamente, a API foi ajustada para suportar o novo fluxo de comunicação, assegurando que os pedidos dos utilizadores sejam encaminhados exclusivamente através do *Manager*. A aplicação Web foi igualmente adaptada, promovendo uma experiência de utilização consistente com a nova arquitetura e reforçando a segu-

rança no acesso aos serviços externos.

Este trabalho resultou numa plataforma coerente, funcional e preparada para servir como base para o desenvolvimento de versões futuras e investigação musical. Para além de garantir a sua manutenção, o trabalho realizado estabelece os alicerces necessários para a integração de modelos avançados de análise musical e para a continuidade do desenvolvimento do ecossistema MERmaid.

1.3 Organização da Dissertação

Capítulo 1: Introdução

Este capítulo introduz o trabalho, enumerando os objetivos e as contribuições do mesmo.

Capítulo 2: Estado da Arte

O capítulo do estado da arte pretende contextualizar o tema, indicar algumas das abordagens e iniciativas mais relevantes no campo do reconhecimento de emoções em música, ao focar as metodologias e técnicas aplicadas e, adicionalmente, explorar projetos relacionados que contribuem para este domínio.

Capítulo 3: Metodologia e Planeamento do Trabalho

Aqui são detalhadas as metodologias, as ferramentas de apoio utilizadas, o planeamento e o ambiente de desenvolvimento que permitiram a realização do projeto.

Capítulo 4: Análise de Sistema

O quarto capítulo apresenta uma análise do sistema MERmaid, principais componentes e principais ações evolutivas tomadas na iteração atual.

Capítulo 5: Implementação

De seguida, é descrito todo o processo de implementação, as ferramentas utilizadas, os módulos desenvolvidos e os desafios superados durante o desenvolvimento.

Capítulo 6: Conclusão

Neste capítulo, realiza-se uma reflexão com base no trabalho realizado e na direção futura para o projeto.

Bibliografia

Na bibliografia, lista-se o conjunto de referências utilizadas e citadas na dissertação.

Apêndice A: Conjunto de Logs do Serviço Manager

Contém o conjunto de *logs* produzidos pelo serviço *manager*.

Apêndice B: Elementos da Página Principal da Interface Web

Compila secções presentes na página inicial da interface Web

Capítulo 2

Estado da Arte

Este capítulo contém uma revisão das principais abordagens e esforços na área de reconhecimento de emoções em música, destacando metodologias e técnicas utilizadas. São ainda referidas e detalhadas algumas ferramentas e projetos similares.

2.1 Reconhecimento de Emoções em Música

O MER tem como objetivo classificar e prever as respostas emocionais transmitidas pela música. Combinando princípios da teoria musical, psicologia e métodos computacionais, procura-se analisar e classificar a relação entre dadas características musicais e as respetivas emoções transmitidas.

O método tradicional de MER pode ser dividido em três etapas (Yang, Dong, & Li, 2018). Na primeira etapa, são estabelecidos os objetivos e o domínio do problema, determinando os formatos musicais (i.e., áudio, letra, outros) e as taxonomias emocionais a adotar. Neste primeiro método está incluída a escolha ou criação do conjunto de dados, ou seja, as músicas e as respetivas anotações emocionais a utilizar, recolhidas tipicamente a partir de voluntários. Segue-se a extração de características para cada uma das instâncias do conjunto de dados, de forma a conseguir resumir cada uma delas a um conjunto de características que as permita distinguir, as chamadas *features*. Um exemplo disso pode ser as batidas por minuto. Por fim, são aplicadas técnicas de aprendizagem automática, de modo a reconhecer padrões que associam as emoções atribuídas pelos anotadores às características musicais. Este processo é, por norma, repetido, variando parâmetros, partições do conjunto de dados e até algoritmos de classificação, com o objetivo de obter um modelo que melhor classifique novas instâncias de emoção desconhecida com base apenas nas suas características.

Como já referido em 1.1, são enfrentados diversos desafios ao trabalhar com MER, seja devido à subjetividade das emoções, ambiguidade nos modelos da psicologia para classificação de emoção (as taxonomias), complexidade de extrair características musicais relevantes usando algoritmos computacionais ou a falta de qualidade e tamanho de conjuntos de dados anotados. Estes obstáculos represen-

tam não apenas barreiras técnicas, mas também oportunidades para exploração científica futura.

Existe já um esforço significativo de investigação em diferentes aspectos do MER (R. Panda et al., 2023): a classificação baseada em *symbolic files*, como o caso do *Musical Instrument Digital Interface* (Interface Digital de Instrumentos Musicais) (MIDI); a classificação *single-label* a partir de excertos de áudio; a classificação *multi-label*; as abordagens dimensionais utilizando regressão; a deteção de variações emocionais ao longo de uma música; a classificação baseada na letra (*lyrics*; as abordagens bimodais/multimodais, seguindo abordagens artesanais; a aprendizagem automática (*machine learning*) ou o *deep learning*, ainda que para tal seja necessária a resolução de diversos problemas e limitações inerentes a cada caso.

2.2 Taxonomias da Emoção

Nos últimos anos, diversos autores observaram que, embora a representação das emoções como um ponto num espaço emocional consiga eliminar a ambiguidade associada às anotações dos modelos categóricos, porque cada ponto num plano é uma emoção distinta, esta abordagem falha em representar a subjetividade humana inerente ao processo de anotação por múltiplos anotadores, já que por norma é resumida ao ponto médio (R. E. S. Panda, 2019). Também é um processo cognitivo mais complexo, complicando ainda mais o processo de anotação já por si difícil.

Esta observação insere-se num debate mais amplo sobre as taxonomias da emoção, abordando métodos sistemáticos de compreensão e categorização das emoções humanas, sendo que, tradicionalmente, estas se encontram divididas em duas abordagens principais: categórica e dimensional.

Modelos categóricos representam as emoções em estados individuais e definidos, utilizando, por exemplo, emoções primárias como alegria, tristeza, medo e raiva definidas por Ekman (Ekman et al., 1987). O diagrama de Hevner, representado na Figura 2.1, exemplifica outra destas taxonomias categóricas, ao apresentar emoções em categorias organizadas em grupos, permitindo visualizar a relação entre diferentes estados emocionais. Apesar de intuitivos e amplamente utilizados, estes modelos enfrentam críticas, devido à rigidez e dificuldade na representação de estados emocionais complexos ou mistos.

Enquanto os modelos categóricos oferecem uma estrutura fixa baseada em categorias ou classes, os modelos dimensionais optam por representar as emoções como um ponto num espaço multidimensional, sendo as dimensões mais conhecidas a valência (positiva ou negativa) e a ativação (alta ou baixa energia), como é o caso do modelo circumplexo de Russell (Russell, 1980), que pode ser observado na Figura 2.2. Embora esta abordagem resolva o problema dos modelos categóricos, que não permitem, por exemplo, distinguir músicas numa mesma categoria, são por outro lado de difícil compreensão e utilização, a que acresce a subjetividade

das respostas emocionais resultante da variabilidade individual na perceção das emoções, que apresenta desafios para alcançar uma representação precisa e universal. Devido a tal facto, é por vezes utilizada uma abordagem híbrida, de forma a complementar ambos os modelos.

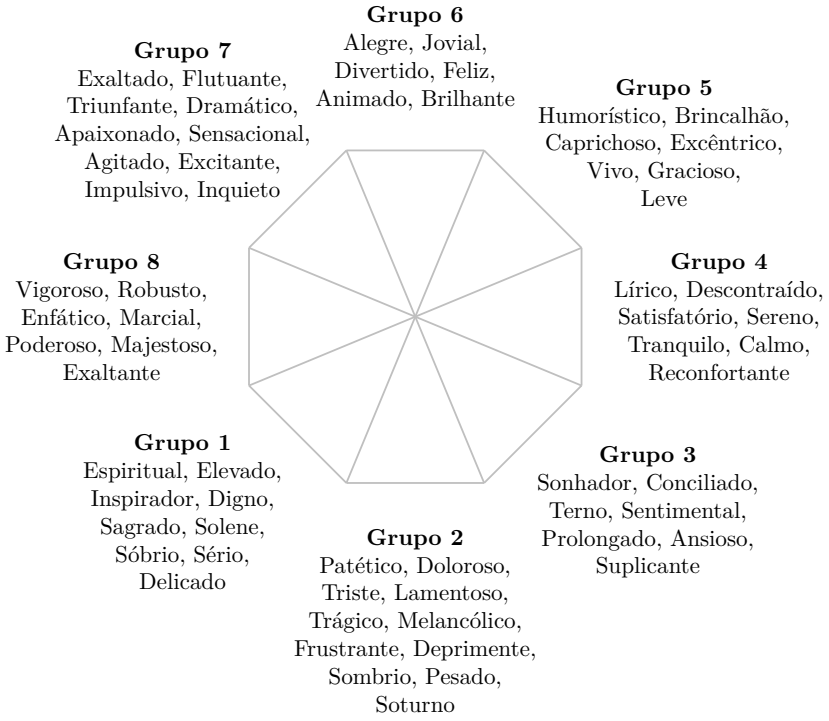

Figura 2.1: Diagrama do Círculo de Hevner com os oito grupos de adjetivos emocionais

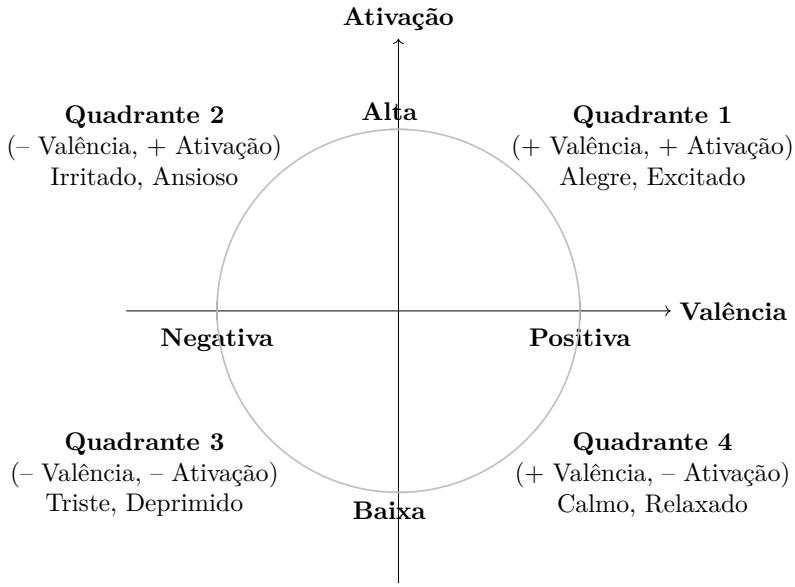

Figura 2.2: Representação dos quadrantes emocionais do Modelo de Russell

Em suma, podemos descrever os modelos categóricos como aqueles que agru-

pam emoções em conjuntos de palavras emocionalmente relevantes. Por outro lado, os modelos dimensionais utilizam pontos distintos num plano multidimensional e baseiam-se em recetores biológicos, considerados diretamente responsáveis pelas respostas emocionais (P. M. L. Louro, 2023).

2.3 Machine Learning e Deep Learning no Contexto MER

Tal como noutras áreas, as técnicas clássicas de *Machine Learning* (ML) dominaram o campo do Reconhecimento de Emoções em Música (MER) durante décadas. No entanto, os avanços têm vindo a estagnar devido à natureza complexa e demorada do processo de criação manual de características áudio emocionalmente relevantes (P. L. Louro et al., 2024). Em termos práticos, muitas das propriedades musicais que sustentam a perceção emocional, incluindo harmonia, ritmo e dinâmica, são de difícil representação através de descritores matemáticos extraídos exclusivamente do sinal áudio. Nos últimos anos, os métodos de *Deep Learning* (DL) ganharam popularidade nesta e noutras áreas, visto serem capazes de aprender automaticamente características relevantes a partir de representações espectrais das músicas.

Os métodos clássicos de ML para MER seguem um processo estruturado (Figura 2.3), que envolve extração e seleção de características, como ritmo ou harmonia, para representar a música numericamente. Algoritmos como *Máquinas de Vetores de Suporte* (SVM), *k-Vizinhos Mais Próximos* (k-NN) e *Floresta Aleatória* são então treinados para associar essas características extraídas do conjunto de músicas de treino a emoções associadas às mesmas. Ainda que amplamente utilizados, a qualidade do modelo resultante depende de todos os passos anteriores, nomeadamente a qualidade do *dataset*, a utilidade das características e o processo de seleção das características extraídas, que tenta eliminar as que apenas introduzem ruído no processo. Este último passo é essencial, porque muitas das características utilizadas foram originalmente desenvolvidas para outras tarefas de análise de áudio, e não para a identificação emocional na música, podendo aumentar apenas a complexidade do modelo ou até reduzir a precisão dos modelos.

Com a adoção de DL em MER e através de técnicas como as *Redes Neuronais Convolucionais* (CNN), que permitem analisar espectrogramas para extrair características, e das *Redes Neuronais Recorrentes* (RNN), que captam a evolução temporal das emoções musicais, tornou-se possível identificar automaticamente padrões emocionais na música sem a necessidade do passo pesado de extração de características (Figura 2.4). O DL representa então um avanço significativo na área, possibilitando um reconhecimento emocional mais eficiente e adaptável a diferentes géneros musicais e contextos. No entanto, observa-se que, na área, continua a existir investigação ativa em ambas as abordagens, bem como em tentativas de as combinar. A engenharia de *features* permite extrair conhecimento explícito sobre a relação entre características musicais e categorias emocionais, ao passo que as redes neuronais profundas, embora mais poderosas, funcionam ainda

como sistemas amplamente opacos do ponto de vista interpretativo.

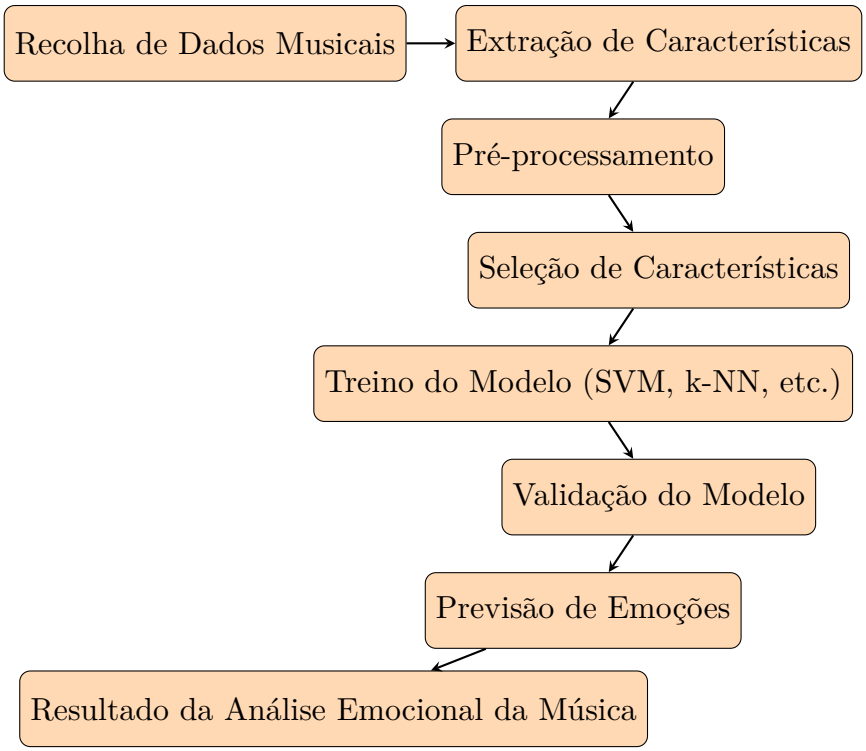

Figura 2.3: Diagrama de processos de Machine Learning para Reconhecimento de Emoção na Música

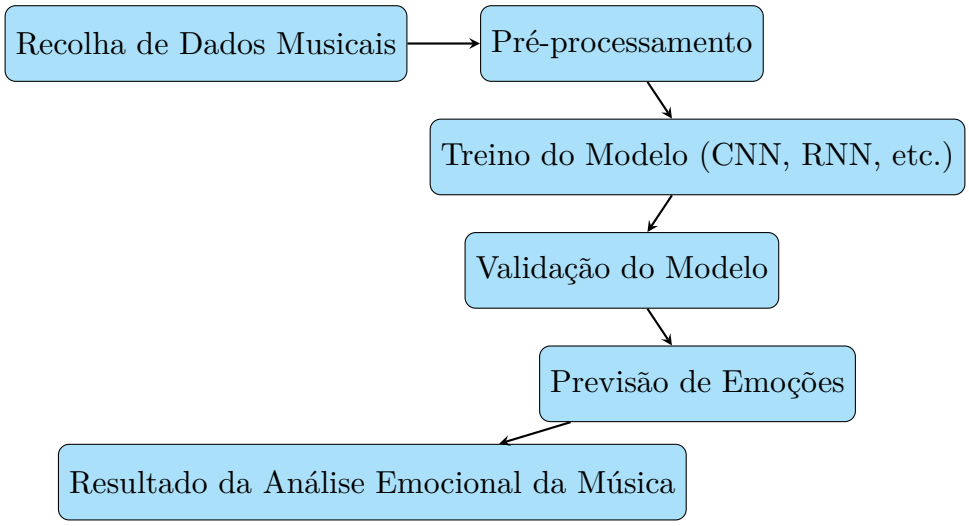

Figura 2.4: Diagrama de processos de Deep Learning para Reconhecimento de Emoção na Música

Ambos os métodos apresentam benefícios e desafios específicos. O ML clássico tem sido amplamente utilizado devido à sua maior interpretabilidade, processos de treino mais leves e à existência de milhares de algoritmos de extração de características desenvolvidos em outros contextos, permitindo aos investigadores uma

análise mais transparente na classificação emocional. No entanto, o seu desempenho está dependente de uma seleção de características, o que pode introduzir subjetividade ou comprometer a generalização para diferentes géneros musicais. Em contraste, o DL elimina essa limitação ao aprender diretamente padrões emocionais através dos dados, mas exige grandes volumes de dados anotados, algo muito difícil de obter na área. Apresenta também elevados requisitos computacionais no processo de treino e enfrenta dificuldades de interpretabilidade, sendo difícil compreender o que influencia as decisões do modelo.

Surge então a possibilidade de utilização de abordagens híbridas, que combinem a extração manual de características com a aprendizagem automática das mesmas. Estudos recentes como o de P. L. Louro et al. representam um avanço crucial para a construção de modelos mais eficazes e adaptáveis.

2.4 Microsserviços e Orquestração

A arquitetura de microsserviços surge como uma evolução das arquiteturas monolíticas, promovendo um modelo de desenvolvimento baseado na divisão de aplicações em serviços independentes e especializados. Cada microsserviço desempenha uma função clara, opera autonomamente e comunica com os restantes através do uso de API ou de mecanismos mais avançados, baseados em eventos como filas de mensagens. Esta abordagem melhora a escalabilidade, a manutenção e o desenvolvimento descentralizado, permitindo que equipas trabalhem em diferentes componentes sem interferências, conseguindo uma independência operacional dos microsserviços e assegurando que falhas isoladas não comprometem o funcionamento global do sistema, aumentando assim a resiliência da infraestrutura.

O sistema MERmaid primitivo executava diversas tarefas distintas e, através da fragmentação em microsserviços e conseguiu garantir escalabilidade e flexibilidade (Figura 2.5). No entanto, tendo em conta o fim académico, alguns serviços eram demasiado pequenos ou específicos e poderiam ter sido integrados noutros semelhantes, reduzindo o número de chamadas entre serviços e melhorando a eficiência global do sistema, sem comprometer a autonomia dos componentes essenciais.

Para a gestão eficiente de um ambiente de desenvolvimento, são necessárias ferramentas de orquestração que garantam a distribuição, a monitorização e a escalabilidade dos serviços. Kubernetes surge como uma solução robusta, oferecendo funcionalidades avançadas, como monitorização detalhada, autoescalonamento, balanceamento de carga e recuperação automática de falhas. Embora estas características sejam vantajosas, a sua configuração inicial complexa e a necessidade de manutenção contínua nem sempre são ideais, especialmente em sistemas de menor dimensão.

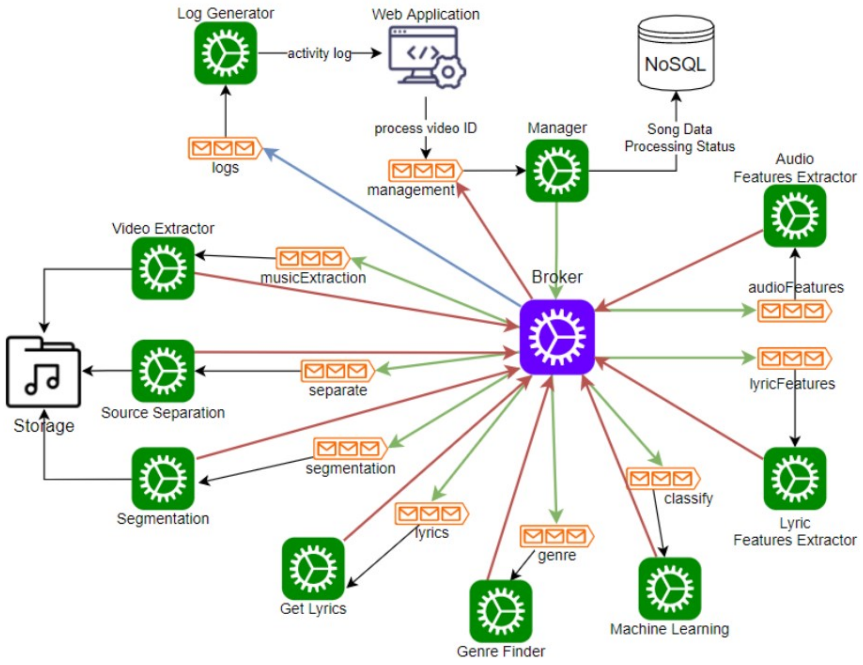

Figura 2.5: Arquitetura geral de microsserviços do sistema MERmaid (António, 2021)

Em contrapartida, o Docker Swarm apresenta-se como uma alternativa mais simples e eficiente, oferecendo orquestração integrada ao ecossistema Docker, o que reduz a sobrecarga operacional e facilita a implementação. Considerando que o MERmaid executa tarefas distintas, mas não necessariamente de alta complexidade, o Docker Swarm revela-se como uma escolha mais adequada, permitindo a distribuição dos serviços sem introduzir complexidade desnecessária, permitindo aos futuros investigadores manter o mesmo operacional.

2.5 Trabalhos Relacionados

O sistema MERmaid não surgiu de forma isolada, sendo o resultado de anos de investigação da equipa. Entre os diversos trabalhos realizados, destacam-se dois projetos de elevada importância: MOODetector (Cardoso et al., 2011) e MOODoke (Grilo & Simões, 2022).

O MOODetector procurou desenvolver uma ferramenta protótipo de *desktop* para a geração automática de *playlists* com base no estado emocional do utilizador (Cardoso et al., 2011). Para tal, seguiu o modelo emocional de Thayer (Figura 2.6), onde cada música é classificada segundo os eixos de valência e ativação. O sistema permite assim criar *playlists* de duas formas: através de uma música de referência (*seed song*), devolvendo músicas com estados emocionais semelhantes, ou através de um trajeto emocional desenhado pelo utilizador, ajustando a seleção musical ao longo do percurso definido.

Figura 2.6: Diagrama do Modelo emocional de Thayer representando os quadrantes emocionais.

Já o MOODoke teve como objetivo desenvolver um sistema capaz de extrair a lírica de vídeos de *karaoke* provenientes da plataforma YouTube, utilizando Optical Character Recognition (Reconhecimento Ótico de Caracteres) (OCR), para depois realizar a classificação emocional do texto obtido (Grilo & Simões, 2022). A adoção de uma abordagem semelhante poderia ser relevante para o MERmaid, tanto na sua versão atual como numa possível iteração futura, através da implementação de um microsserviço *speech-to-text* lírico, permitindo a extração de letras de músicas de forma automatizada, simulando OCR mas aplicado ao áudio.

No âmbito dos avanços proporcionados pelos trabalhos anteriores, surgiu em 2019 o primeiro protótipo MERmaid, com o objetivo de demonstrar o conceito MER ao público em geral e procurando criar uma ferramenta de apoio para os investigadores. Além destes contributos diretos, é essencial reconhecer diversos outros alunos envolvidos em componentes posteriormente integrados no sistema, que desempenharam um papel essencial na construção da infraestrutura, evolução dos modelos de classificação e microsserviços presentes, permitindo um constante desenvolvimento e expansão.

Ao longo da evolução do projeto, diversos estudantes tiveram um papel determinante na consolidação das bases conceptuais e tecnológicas que sustentam o MERmaid. A primeira versão funcional, desenvolvida em 2019, contou com o contributo de Tiago Areias e Tiago António, então estudantes de Licenciatura em Engenharia Informática (LEI), que implementaram a camada Web e a primeira API em React, estabelecendo a interface inicial do sistema. Em paralelo, Ricardo António explorou o paradigma de microsserviços aplicados ao reconhecimento de emoções na música, criando um protótipo composto por três serviços fundamentais: obtenção de letras, extração experimental de um conjunto reduzido de *features* e classificação emocional preliminar. Estes esforços constituíram o primeiro alicerce técnico concreto do projeto MERmaid.

Em 2021, o trabalho avançou para o nível de mestrado, com Tiago António e João Canoso realizando uma reestruturação significativa do sistema. Nesta fase, Tiago António aprofundou a investigação MER, avaliando múltiplas bibliotecas de extração de características áudio, treinando modelos supervisionados e realizando estudos de seleção de *features*. Simultaneamente, João Canoso dedicou-se à concepção de uma infraestrutura de computação avançada para o MERmaid, propondo uma solução baseada em Kubernetes e mecanismos de orquestração distribuída. Embora esta versão não tenha incluído a atualização da aplicação Web, deixou um conjunto de propostas tecnológicas que viriam a influenciar iterações subsequentes.

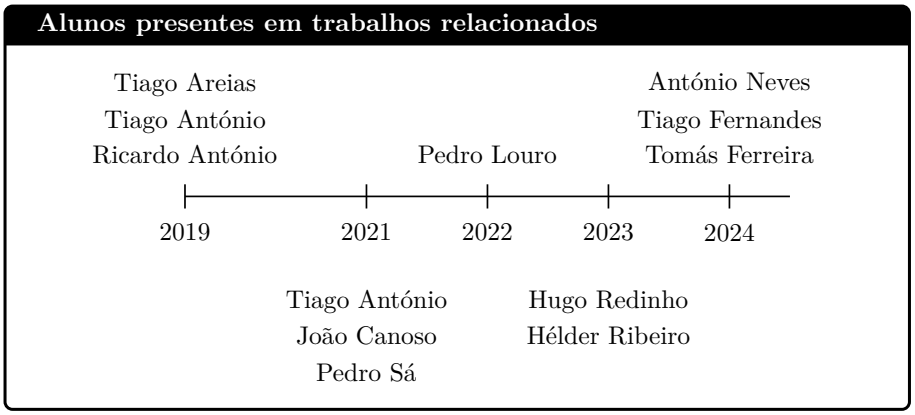

Figura 2.7: Representação temporal dos diversos alunos com trabalhos relacionados com o Projeto MERmaid

Os anos seguintes envolveram contributos associados ao CISUC, através dos trabalhos de Pedro Sá, Pedro Louro, Hugo Redinho e Tomás Ferreira, cuja investigação incidiu sobretudo no desenvolvimento de novas características musicais, abordagens bimodais e metodologias de aprendizagem profunda aplicadas à classificação emocional. Estes trabalhos, maioritariamente de natureza científica, contribuíram para o avanço teórico da área e forneceram modelos e ideias que poderão futuramente ser integrados no ecossistema MERmaid. O percurso evolutivo incluiu ainda a intervenção de Hélder Ribeiro, responsável pela criação da iteração inicial da versão Web atualmente em uso, partindo de um esboço conceptual deixado por João Canoso. Este desenvolvimento foi posteriormente continuado e expandido por António Neves e Tiago Fernandes, também estudantes de licenciatura, que refinaram a interface e aprofundaram a integração dos serviços de *backend*.

Em conjunto, estes contributos moldaram não apenas a linha temporal apresentada na Figura 2.7, mas também a própria identidade do projeto, cuja evolução reflete a natureza multidimensional entre engenharia de software, investigação musical e aprendizagem automática. Cada iteração expandiu o alcance conceptual e tecnológico do MERmaid, criando a base sobre a qual as versões seguintes foram projetadas e implementadas, incluindo a que é apresentada nesta dissertação.

2.6 Balanço Histórico e Tecnológico

A revisão do estado da arte permitiu consolidar uma compreensão abrangente do domínio do MER, destacando tanto o potencial como as limitações atuais das abordagens existentes. A análise das taxonomias emocionais, das estratégias de extração de características e dos métodos de ML e DL revelou que persistem os desafios fundamentais da área: a subjetividade das emoções, a escassez de *datasets* anotados e a dificuldade em representar computacionalmente aspectos complexos da música. Ao mesmo tempo, ficou evidente que as tendências mais recentes privilegiam abordagens baseadas em *deep learning*, que eliminam a necessidade de extração manual de *features* e apresentam tempos de inferência reduzidos. No contexto de um protótipo integrado como o MERmaid, esta constatação é particularmente relevante, justificando a adoção de técnicas modernas de separação de fontes, segmentação e transcrição automática como etapas intermédias que complementam os modelos emocionais, à semelhança do que vem sendo investigado no âmbito do projeto MERGE [1].

Por outro lado, a análise histórica das diferentes versões do MERmaid permitiu identificar um conjunto consistente de lições práticas. A primeira iteração demonstrou a viabilidade técnica do conceito e estabeleceu as bases para uma arquitetura modular. A segunda fase, embora ambiciosa, evidenciou um excesso de complexidade ao propor uma infraestrutura orientada para cenários industriais, pouco compatível com o contexto académico e com os recursos disponíveis. Esta reflexão motivou a abordagem seguida na presente dissertação: reduzir a fragmentação, evitar sobrecarga de orquestração e privilegiar uma arquitetura pragmática baseada em Docker, com microsserviços isolados, mas funcionais e facilmente combináveis através do *Manager*. Esta orientação coloca o foco naquilo que é realmente essencial para a investigação em MER: disponibilizar um conjunto de componentes estáveis, reutilizáveis e extensíveis, prontos para acolher os modelos atualmente em desenvolvimento.

[1] `https://doi.org/10.54499/PTDC/CCI-COM/3171/2021`

Capítulo 3

Metodologia e Planeamento do Trabalho

Este capítulo detalha as metodologias e ferramentas de apoio utilizadas, o planeamento e o ambiente de desenvolvimento criado para o trabalho desenvolvido.

3.1 Ferramentas de Apoio e Controlo de Versões

O desenvolvimento do projeto beneficiou significativamente da utilização ativa de ferramentas que suportam a colaboração, a organização e o controlo de versões de código. Entre as principais destacam-se o GitHub[1], plataforma de gestão de desenvolvimento, e o Obsidian[2], utilizado como sistema de apoio à documentação, anotações técnicas e organização de ideias.

3.1.1 GitHub como Plataforma de Desenvolvimento Colaborativo

O uso da ferramenta GitHub permitiu o armazenamento centralizado do código-fonte, o acompanhamento das diferentes iterações através de *issues*, *pull requests* e do GitHub Projects; e a integração contínua dos diversos componentes do sistema, criando um fluxo de trabalho contínuo e controlado.

A estrutura de repositórios foi organizada de forma modular para refletir a arquitetura de microsserviços do sistema, permitindo assim que cada repositório corresponda a um serviço ou componente independente, o que facilita a manutenção e a escalabilidade do projeto.

Os repositórios foram organizados em três grandes grupos: os serviços principais, responsáveis pelas tarefas de processamento de áudio e classificação emo-

[1] `https://www.github.com`
[2] `https://obsidian.md`

cional, como `SourceSeparation`, `LyricsRetrieval` ou `EmotionClassification`; a infraestrutura e orquestração, que inclui configurações Docker, *scripts* de automatização e *pipelines*, como `Manager` e `dev-orchestrator`; e as interfaces e componentes de `front-end`, nomeadamente a aplicação Web (`mermaid-fe`) e a API que faz a ponte entre esta e os microsserviços (`mermaid-bk`). Esta estrutura modular facilitou o isolamento das dependências, a reutilização de componentes e o desenvolvimento independente de cada serviço.

Estratégia de Branching

A estratégia de *branching* adotada foi simples mas eficaz. A `main` mantém a versão estável do sistema, sempre pronta para um ambiente de produção. A `development` serve como espaço intermédio para integrar novas funcionalidades e correções antes da sua fusão com a versão estável. Finalmente, os ramos do tipo `feature/`*`nome`* são criados para o desenvolvimento de funcionalidades específicas e, posteriormente, integrados através de *pull requests*. Esta abordagem reduz conflitos, permite a revisão de código e mantém um ciclo de desenvolvimento claramente organizado.

Padronização de Commits

Para garantir consistência semântica e legibilidade histórica no desenvolvimento da interface (`mermaid-fe`) e da API (`mermaid-bk`), foram definidas regras padronizadas para mensagens de *commit*. Estas regras, aplicadas automaticamente através do ficheiro de configuração `commitlint.config.js`, asseguram que cada alteração ao repositório siga uma taxonomia clara e que as mensagens mantenham uma estrutura previsível e informativa, como pode ser observado na Tabela 3.1.

Tipo	**Descrição**
`feat`	Introdução de uma nova funcionalidade no sistema.
`fix`	Correção de um erro ou comportamento inesperado.
`docs`	Alterações exclusivamente na documentação, sem impacto funcional.
`style`	Mudanças de formatação ou estilo que não alteram o comportamento (ex.: indentação, espaços).
`refactor`	Reestruturação interna do código sem modificar funcionalidades externas.
`perf`	Alterações orientadas a melhorar o desempenho.
`test`	Adição ou modificação de testes automatizados.
`build`	Alterações que afetam o sistema de build ou dependências externas.
`ci`	Modificações no *pipeline* de integração contínua ou configurações associadas.
`chore`	Tarefas de manutenção que não alteram o código executável (ex.: limpeza de ficheiros, ajustes de configuração).
`revert`	Reversão explícita de um commit anterior.

Tabela 3.1: Tipos de mensagens de commit adotados

Esta padronização facilita a revisão de código e reduz ambiguidades, algo particularmente importante num projeto de investigação com múltiplos intervenientes

ao longo do tempo.

Metodologia e Gestão de Tarefas

Através da ferramenta GitHub Projects, tornou-se possível gerir *issues* (tarefas, melhorias e correções), sendo cada tarefa associada a um repositório e respetivo *branch*, bem como ao estado atual da mesma: *Planned*, *Ready*, *In Progress*, *In Review*, *Done*. Foi ainda possível associar *sprints*, tamanhos a cada tarefa (XS, S, M, L, XL) e prioridades (P0, P1, P2), facilitando o desenvolvimento contínuo ao evitar a perda de foco nos objetivos.

O trabalho foi desenvolvido de forma iterativa, como pode ser observado na Figura 3.1, seguindo uma abordagem ágil inspirada nas metodologias Scrum, mas adaptada à natureza académica do projeto. Não foram implementadas todas as práticas formais do Scrum, como os papéis de *Product Owner* ou *Scrum Master*, mas manteve-se o princípio de ciclos curtos de desenvolvimento e revisão contínua.

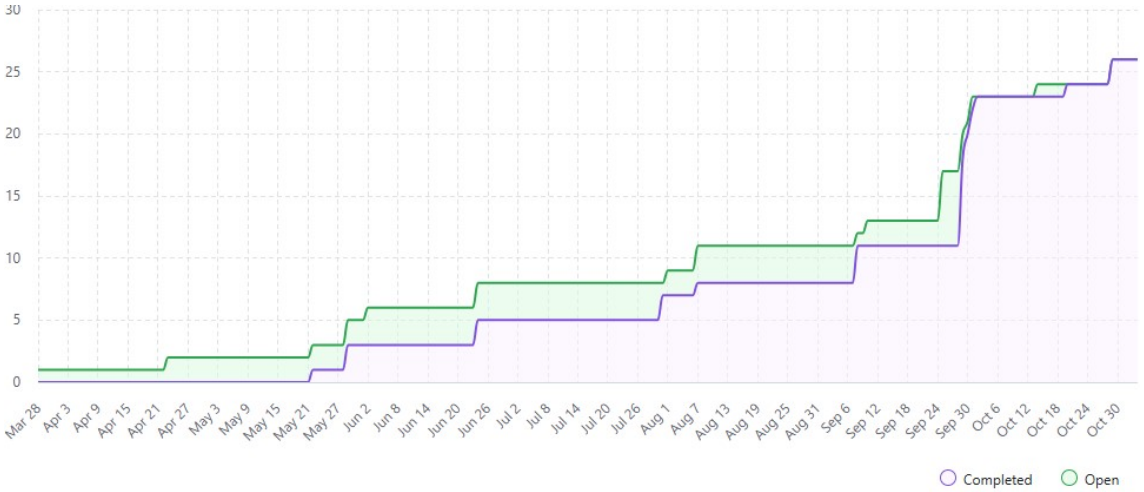

Figura 3.1: BurnUpChart correspondente às tarefas concluídas em contraste às por concluir ao longo do período de desenvolvimento

Cada *sprint*[3] tinha uma duração aproximada de duas semanas, com reuniões semanais com o orientador para avaliação do progresso e redefinição de prioridades. Esta estrutura permitiu uma evolução contínua do sistema, com entregas incrementais e ajustes frequentes às necessidades identificadas.

3.1.2 Obsidian como Ferramenta de Documentação

O Obsidian desempenhou um papel complementar, funcionando como um espaço de documentação dinâmica e pessoal. Através da criação de notas interligadas

[3] *Sprint*: ciclo de trabalho

(Figura 3.2) e da utilização da linguagem *Markdown*[4], foi possível: registar decisões técnicas e metodológicas tomadas; documentar erros e soluções adotadas; reunir referências bibliográficas e resumos de artigos científicos antes da importação para o Mendeley[5]; manter uma linha temporal de progresso e reflexão sobre o projeto; e detalhar dúvidas ou considerações em preparação para as reuniões com o orientador.

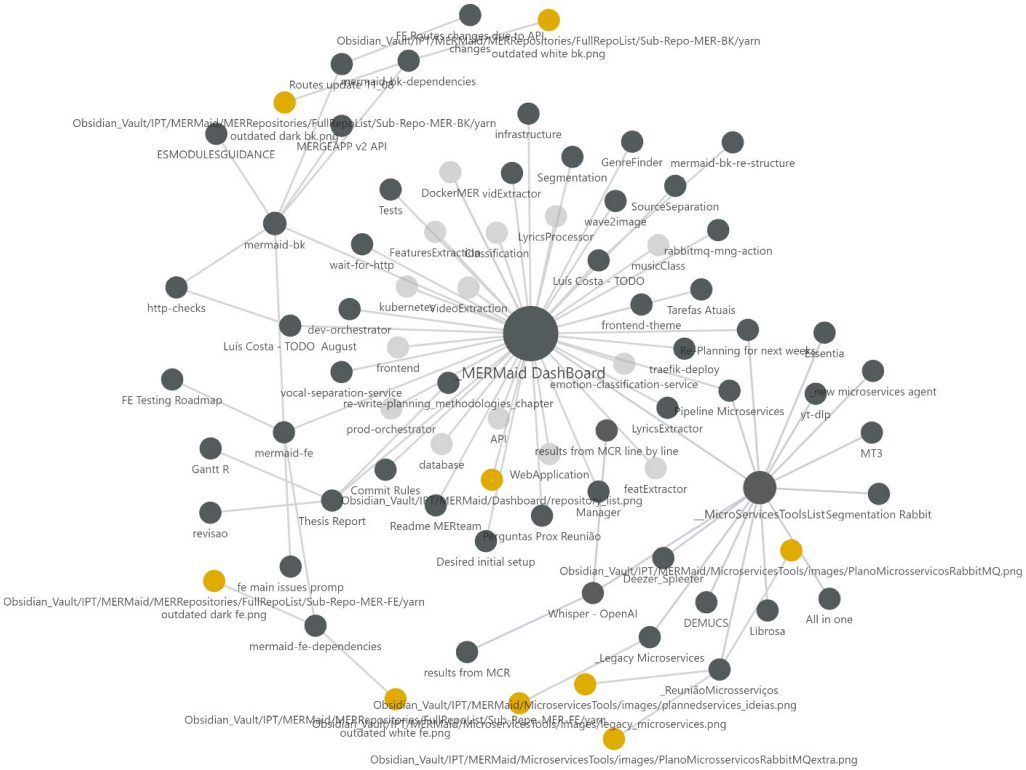

Figura 3.2: Visualização das notas interligadas no Obsidian

O uso em paralelo do GitHub e do Obsidian permitiu um desenvolvimento consistente ao juntar uma ferramenta de controlo técnico a uma ferramenta de apoio cognitivo. Foi ainda utilizada a ferramenta Notion[6] como apoio na organização de referências e ideias, mas rapidamente substituída pelo Mendeley, aplicação para a organização de referências que, em conjunto com o Notion, permitiu uma organização mais sucinta.

3.2 Ambiente de Desenvolvimento

Na fase inicial do trabalho, foi proposta a construção de um ambiente de desenvolvimento que permitisse consistência, segurança e conveniência, através da

[4] *Markdown* é uma linguagem simples para formatação de texto
[5] `https://www.mendeley.com/`
[6] `https://www.notion.com`

utilização do Windows Subsystem for Linux 2 (WSL2) integrado com Docker, Dev Containers e Visual Studio Code (VS Code).

A utilização do Windows nativo para vários projetos pode apresentar diversos inconvenientes, especialmente ao utilizar ferramentas como Node.js, Python, Ruby, Docker ou até mesmo bases de dados, desde problemas com pacotes, dependências, execução do Docker, ou variados problemas de *deploy*, como pode ser observado na Secção 3.2.

Critério	**Windows (nativo)**	**WSL2 (Linux Subsystem)**
Compatibilidade com pacotes	Problemas frequentes com dependências em *npm*, *pip*, etc.	Elevada compatibilidade, por operar num ambiente Linux real
Coerência com ambiente de produção	Possíveis divergências (ex.: caminhos, case sensitivity)	Ambiente idêntico ao de servidores Linux
Desempenho com Docker	Execução lenta, dependente de máquina virtual	Execução direta no kernel Linux, mais eficiente e leve
Integração com ferramentas	Boa integração com software Windows, mas com limitações Unix	Integração fluida com VS Code, mantendo acesso a apps Windows
Gestão de dependências	Sujeito a conflitos e erros de compilação	Geridas nativamente em Linux, reduzindo falhas

Tabela 3.2: Comparação entre desenvolvimento nativo em Windows e ambiente WSL2

Em suma, o WSL2 executa um ambiente Linux completo em segundo plano, sem a necessidade de utilização de uma máquina virtual independente ou `dual booting`, oferecendo assim maior velocidade e compatibilidade.

Embora não seja utilizado Ruby no projeto, o seguimento da presente secção de configuração resulta de uma típica instalação de Ruby on Rails no Windows 11 [7], alterada para refletir apenas as necessidades do ambiente desejado, evidenciando a versatilidade do WSL2.

3.2.1 Configuração do Ambiente de Desenvolvimento

Para garantir um ambiente de desenvolvimento coerente com o de produção, foi utilizado o WSL2, que permite executar uma distribuição Linux nativa dentro do Windows. Esta secção documenta os principais passos seguidos para a sua configuração, com o intuito de facilitar a reprodutibilidade do mesmo. Para a

[7] https://gorails.com/setup/windows/11

solução presente é essencial uma versão igual ou superior ao Windows 10 (version 2004, Build 19041 and higher) ou Windows 11.

Instalação do WSL2 e da Distribuição Ubuntu

Antes de iniciar a instalação da distribuição, é recomendado confirmar que o WSL2 se encontra atualizado, de modo a evitar incompatibilidades ou limitações de desempenho.

Seguidamente realiza-se a instalação da distribuição Linux desejada, neste caso Ubuntu, podendo ser executada de forma simples através da linha de comandos:

```
1 wsl --update # atualizar para a versão mais recente
2 wsl --install -d Ubuntu # instalar a distribuição desejada
```

Excerto 3.1: Instalação da distribuição Ubuntu no WSL2

Na eventualidade de serem necessárias múltiplas distribuições, devem ser aplicados os seguintes comandos, não necessários ao projeto atual, embora úteis para gerar ambientes similares:

```
1 wsl --list -v       # listar as distribuições locais
2 wsl --list --online # listar as distribuições disponíveis
3
4 wsl --install -d <distro name> # instalar a distribuição desejada
5 wsl -d <distro name> # definir a distribuição predefinida
```

Excerto 3.2: Instalação de distribuições adicionais no WSL2

Foi ainda instalada a ferramenta Windows Terminal, no Windows, visto facilitar a seleção de ambientes e consequente utilização de comandos, tornando-se este o terminal predefinido.

Atualização do Sistema e Instalação de Dependências

De seguida, foi essencial preparar o sistema com as ferramentas essenciais, recorrendo-se então ao gestor de pacotes Advanced Packaging Tool (APT) para instalar e atualizar um conjunto de bibliotecas fundamentais ao desenvolvimento:

```
1 sudo apt update -y  # Atualizar a lista de pacotes disponíveis
2 sudo apt upgrade -y # Atualizar os pacotes instalados
3 sudo apt install build-essential rustc libssl-dev libyaml-dev zlib1g-dev
    libgmp-dev -y   # Instalar dependências
```

Excerto 3.3: Atualização de pacotes e instalação de dependências

Configuração do Git e Identidade do Utilizador

Para um uso apropriado do controlo de versões é necessária a configuração do `git`, procedendo-se à inserção do nome e e-mail como ferramentas de identificação:

```
# Definir o nome de utilizador global para os commits
git config --global user.name "YOUR NAME"

# Definir o e-mail associado à conta Git (identificação dos commits)
git config --global user.email "YOUR@EMAIL.com"
```

Excerto 3.4: Configuração do Git

Se desejado, pode também ser omitido o comando `--global`, sendo criada uma configuração local e não global.

Autenticação SSH para GitHub

Após a configuração do utilizador, foi necessário estabelecer um método de autenticação seguro entre o ambiente local e o GitHub, recorrendo-se a chaves SSH. A utilização do protocolo Secure Shell (SSH)[8] permitiu uma autenticação e comunicação encriptadas, garantindo a integridade e a segurança nas interações do repositório remoto com o ambiente local:

```
# Gerar um novo par de chaves SSH
ssh-keygen -t ed25519 -C "YOUR@EMAIL.com"

# Nota: Nunca partilhar a chave pessoal!

# Mostrar a chave pública no terminal
cat ~/.ssh/id_ed25519.pub

# Copiar o valor apresentado e adicioná-lo à conta GitHub:
#   GitHub → Settings → SSH and GPG keys → New SSH Key
```

Excerto 3.5: Configuração do Git e Autenticação SSH

Autenticação SSH para VS Code

Ao utilizar o VS Code em ambientes de Dev Containers, é essencial garantir que as chaves de autenticação SSH configuradas no sistema anfitrião estejam acessíveis dentro do *container*. Para tal, recorreu-se à ferramenta `keychain`, que permite gerir e conservar as chaves SSH entre sessões, evitando a necessidade de reintroduzir manualmente a chave a cada utilização do sistema:

[8] `https://linuxcommand.org/lc3_man_pages/ssh1.html`

```
# Instalar a ferramenta Keychain
sudo apt install keychain -y

# Adicionar o Keychain ao ficheiro .bashrc para um início automático
echo 'eval "$(keychain --eval --agents ssh id_ed25519)"' >> ~/.bashrc

# Nota: se foi utilizada uma chave com nome diferente (por exemplo, id_rsa),
# deve substituir-se "id_ed25519" pelo nome correspondente.

# Recarregar o .bashrc para aplicar as alterações na sessão atual
source ~/.bashrc
```

Excerto 3.6: Configuração e Autenticação SSH para VS Code

Gestão de Recursos do WSL2

Esta secção é opcional, mas recomendada para máquinas com recursos limitados, pois permite uma utilização mais confortável do WSL2.

Deve ser criado um ficheiro `.wslconfig` no diretório pessoal do Windows e realizada a alteração do conteúdo do mesmo, consoante os recursos disponíveis e a necessidade do utilizador:

```
# C:\Users\YourUsername\.wslconfig

[wsl2]
memory=4GB   # Limita a utilização de memória RAM
processors=2 # Limita a utilização de núcleos de CPU
swap=2GB     # Limita a memória virtual
localhostForwarding=true  # Permite redirecionamento de portas localhost entre Windows e WSL2
kernelCommandLine=sysctl.vm.drop_caches=1  # Otimiza a gestão de cache de memória
experimentalMemoryReclaimer=true  # Ativa libertação automática de memória não utilizada
```

Excerto 3.7: Conteúdo do ficheiro .wslconfig para controlo de recursos

Capítulo 4

Análise de Sistema

Neste capítulo descreve-se o sistema MERmaid na sua composição anterior, enumerando os diversos repositórios de componentes e introduzindo os serviços, bem como a evolução e direção tomada na iteração atual.

4.1 Arquitetura do Sistema MERmaid

Embora exista um aumento no interesse de estudos MER, a falta de um sistema dedicado ao reconhecimento emocional não estimula esta área de investigação. Assim, o projeto MERmaid surgiu na sequência da necessidade de um sistema que permita, dada uma música, saber qual a emoção transmitida associada. Este pode ser compilado em duas grandes versões, representadas na Figura 4.1, a versão protótipo de (Areias & António, 2019) e MERmaid Web App,

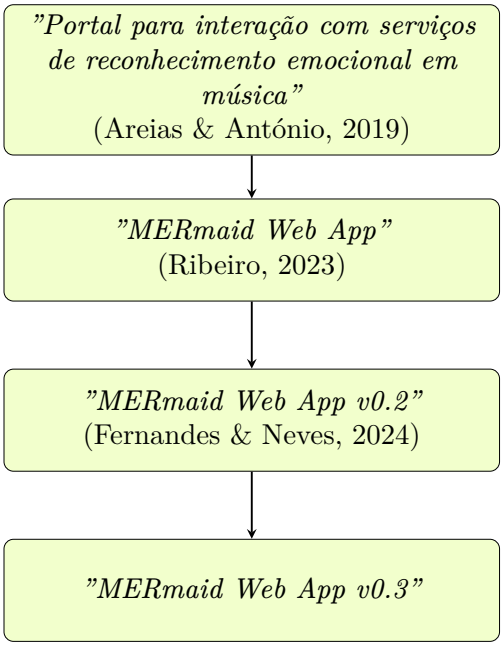

Figura 4.1: Evolução dos Trabalhos MERmaid

A versão anterior do sistema MERmaid apresentava uma estrutura mais reduzida, formada pelos elementos essenciais à comunicação entre o utilizador e os serviços de processamento. A plataforma adotou uma arquitetura baseada em microsserviços, essencialmente composta por uma interface Web, construída com o uso das ferramentas React, Bootstrap e D3.js, uma API desenvolvida em Express.js, uma infraestrutura de suporte baseada em RabbitMQ, para comunicação assíncrona, PostgreSQL para armazenamento de dados e WebSockets para atualizações em tempo real, como se verifica na Figura 4.2.

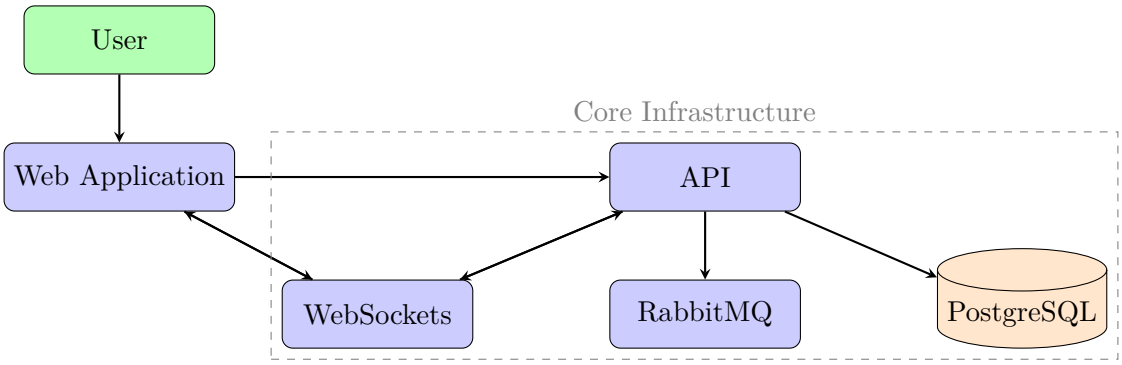

Figura 4.2: Arquitetura Geral do Sistema MERmaid v0.2

À medida que o projeto evoluiu, tornou-se necessária a criação de serviços especializados, como a extração de áudio do YouTube; a segmentação de áudio; a extração de letras; a separação vocal; a criação de uma onda musical; a classificação de género musical e a classificação emocional, cada um responsável por uma etapa distinta do fluxo de processamento. Estes microsserviços não faziam parte da arquitetura na versão anterior do MERmaid, sendo integrados na presente iteração.

Deste modo, a arquitetura foi naturalmente expandida, passando a incluir um maior número de microsserviços. Esta evolução, através da integração de um serviço *Manager*, permitiu a independência entre microsserviços e preparou o sistema para o volume crescente de funcionalidades e para futuras iterações.

Atualmente, a plataforma adota a mesma base tecnológica, utilizando o RabbitMQ, como intermediário central para a comunicação entre componentes, e uma infraestrutura composta por: uma interface Web, uma API, que faz uso de WebSockets para atualizações em tempo real; duas bases de dados, sendo os dados dos utilizadores guardados em PostgreSQL e os dados de processamento em MongoDB; um serviço *Manager*, que controla o fluxo de ações recebidas no RabbitMQ e envia os pedidos aos serviços de processamento desenvolvidos em Python e Node.js. Toda esta informação encontra-se compilada sob a forma de diagrama na Figura 4.3.

No domínio de DevOps, a aplicação encontra-se containerizada com Docker e Docker Compose, com possível escalonamento para Docker Swarm ou Kubernetes.

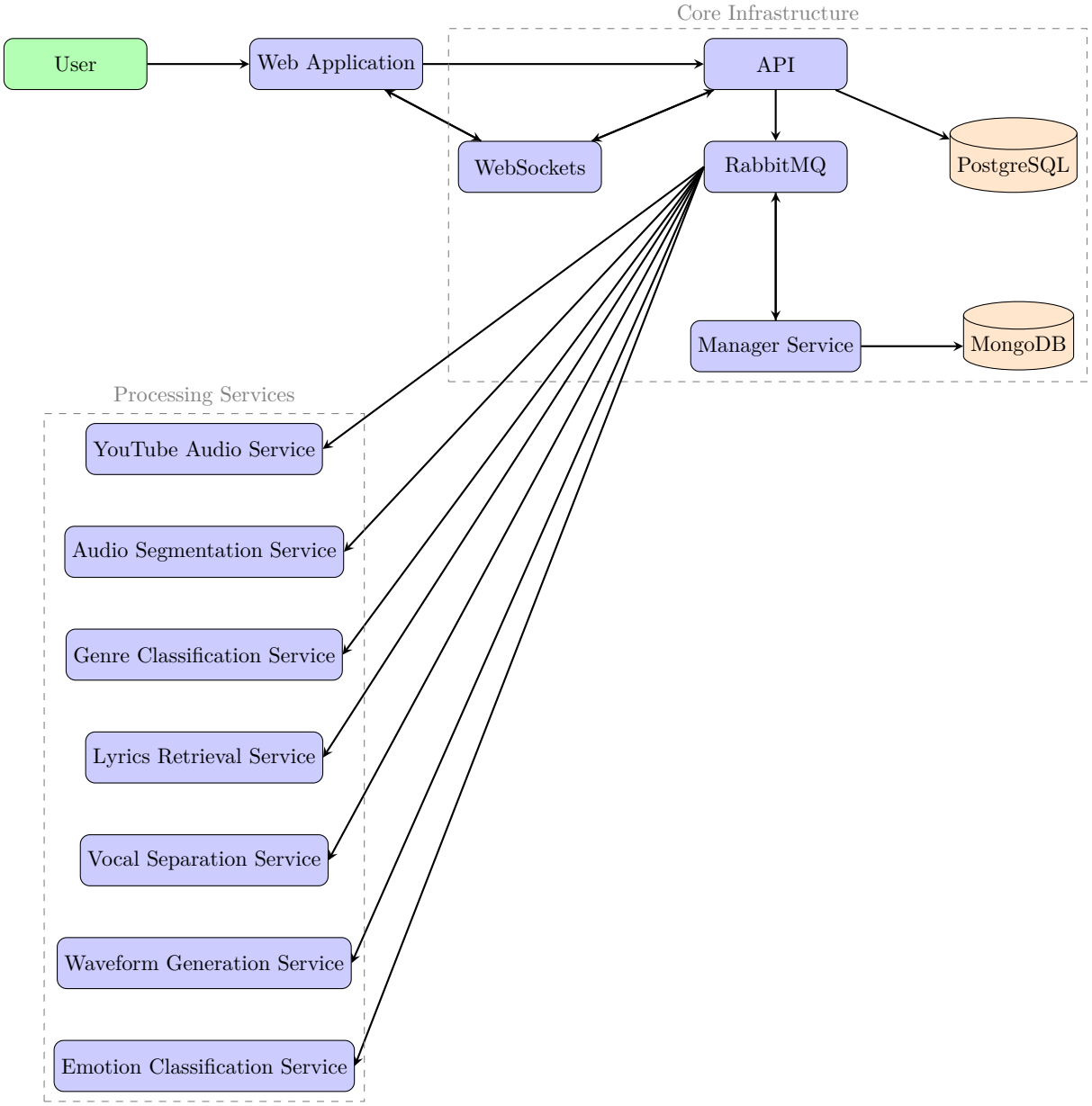

Figura 4.3: Arquitetura Geral Atualizada do Sistema MERmaid

Assim, como já mencionado, este projeto compila o trabalho de diversos membros ao longo dos anos, do que resulta uma elevada quantidade de repositórios de componentes e serviços. No início do desenvolvimento, o projeto constava de 31 repositórios: 15 ativos, 4 planeados e 12 arquivados. Atualmente, o mesmo apresenta 37 repositórios: 16 ativos, 3 planeados e 18 arquivados, como consta na Tabela 4.1.

É ainda possível confirmar a existência de uma extensa lista de microsserviços na organização, que, embora não integrados, serviram como provas de conceito, testes ou apenas estiveram presentes nas versões iniciais do projeto MERmaid. Ainda assim, foi essencial analisar cada um destes *legacy services*, aproveitando os mesmos sempre que possível.

Repository	Description	Status
audio-processing-pipeline	Repository for Local Pipeline Testings	Active
audio-segmentation-service	Service to Segment a Given Audio Using Multiple Algorithms	Active
dev-orchestrator	Docker Compose Files and Git Submodules for Development	Active
frontend-theme	Project Frontend Theme Based on Template	Active
GenreFinder	Service to Detect Music Genre	Active
lyrics-retrieval-service	Service to Retrieve Lyrics from a Given Audio	Active
mermaid-bk	WebAPI for the MERmaid System Web App	Active
mermaid-fe	Frontend for the MERmaid System Web App	Active
pipeline-manager-service	Service for Managing the Audio Processing Pipeline	Active
rabbitmq-mng-action	GitHub Action for RabbitMQ Management	Active
source-separation-service	Service to Separate Vocals from Music	Active
Tests	Testing for the Various Components	Active
traefik-deploy	Traefik for Staging Deployment	Active
wait-for-http	Utility Service for Waiting on HTTP Endpoints	Active
wave2image	Service for Generating Waveform Images Based on a Given Audio	Active
youtube-downloader-service	Service to Retrieve Audio from YouTube	Active
emotion-classification-service	Service for Emotion Classification (Deep Learning)	Planned
infrastructure	Infrastructure as Code for Deployment	Planned
prod-orchestrator	Production Deployment Orchestration	Planned
API	Legacy API Implementation	Archived
Classification	Legacy Service for Classification	Archived
database	Legacy Database Implementation	Archived
DockerMER	Legacy Provisioning	Archived
featExtractor	Legacy Service for Extracting Audio Features	Archived
FeaturesExtraction	Legacy Service for Features Extraction	Archived
frontend	Legacy Frontend Implementation	Archived
kubernetes	Legacy Deployment Definitions using Kubernetes	Archived
LyricsExtractor	Legacy Service to Retrieve Lyrics from Genius	Archived
LyricsProcessor	Legacy Service for Processing Retrieved Lyrics	Archived
Manager	Legacy Management Service to Manage the Flow of Architecture	Archived
Microservices-tryouts	Repository for Microservice Learning and Testings	Archived
musicClass	Legacy Music Classification Service	Archived
Segmentation	Legacy Service to Segment a Given Audio to N Given Segments	Archived
SourceSeparation	Legacy Service to Separate the Audio Sources (Accompaniment/Wave etc.)	Archived
VideoExtraction	Legacy Service for Video Extraction	Archived
vidExtractor	Legacy Service to Retrieve Audio from YouTube	Archived
WebApplication	Legacy Web Application	Archived

Tabela 4.1: Lista de repositórios da organização MER-team

4.2 Microsserviços

Como já mencionado, o MERmaid segue uma arquitetura de microsserviços, apresentando, portanto, diversos microsserviços com funções claras e essenciais ao funcionamento do projeto. Nesta secção encontram-se os principais serviços que ajudam nas tarefas de recolha e processamento de dados, o contexto de cada um, as ferramentas que os destacam e as principais alterações realizadas. Deve-se ainda ter em conta que esta é uma introdução aos serviços e ferramentas, como eles existiam e os problemas encontrados, sendo a direção atual explorada em profundidade no Capítulo 5.

4.2.1 YouTube Downloader

Visto ser essencial obter ficheiros de áudio para reconhecimento de emoções, foi necessário criar, em iterações anteriores, um serviço de recolha de dados, neste caso através da API da plataforma YouTube e de um *YouTube Downloader*, permitindo ao utilizador apenas mencionar o vídeo desejado ao fazer o pedido de processamento.

A ferramenta de linha de comandos YouTube-DL[1] permite, dado um link, fazer o download do ficheiro áudio do respetivo vídeo da plataforma YouTube. O projeto MERmaid utilizava esta ferramenta como microsserviço, recebendo pedidos através do RabbitMQ e devolvendo ficheiros áudio `.mp3` dos respetivos vídeos prontos a serem processados.

O YouTube-DL é um projeto *open-source* (código aberto) que, de momento, se encontra sem manutenção ativa. Estas ferramentas devem ser atualizadas de forma a acompanhar a evolução contínua do YouTube, evitando possíveis falhas ou contornando novas barreiras impostas pelo mesmo.

Tal situação foi observada no sistema MERmaid, nem sempre sendo possível obter o ficheiro `.mp3` ao realizar um pedido, o que impede todo o processamento subsequente. Assim, foi ponderado o YouTube-DLP[2], projeto *fork* do YouTube-DL, visto ser regularmente atualizado e acompanhado de extenso apoio da comunidade.

Este apoio permitiu ao YouTube-DLP obter novas qualidades: melhoria no desempenho e eficiência na extração; extração de outros sites que não o YouTube, expandindo as suas possibilidades; atualizações frequentes, acompanhando necessidades e alterações nos sites dos quais extrai; e receber novas funcionalidades, consoante o esforço da comunidade.

Assim, e considerando as necessidades do MERmaid, que englobam desde a manutenção do sistema ao desejo futuro de interação com outras plataformas de música, o YouTube-DLP tornou-se a escolha mais adequada à tarefa. Desta forma,

[1] https://github.com/ytdl-org/youtube-dl
[2] https://github.com/yt-dlp/yt-dlp

a solução ideal foi proceder a uma alteração de ferramentas, criando um microsserviço atualizado e mais robusto, facilitando, assim, novas integrações nas iterações futuras do projeto.

4.2.2 Source Separation

A Source Separation (separação de fontes sonoras) de um dado áudio consiste no processo de isolar diferentes componentes: voz, instrumentos e/ou ruído ambiente, recuperando individualmente as fontes originais presentes numa gravação, e permitir assim uma análise mais detalhada de cada elemento sonoro, apresentando diversas aplicações na análise, edição e manipulação de dados provenientes de áudio (Virtanen, 2007).

No contexto MER, a separação de fontes sonoras assume um papel fundamental, permitindo isolar componentes como a voz ou o acompanhamento instrumental. Esta técnica procura melhorar a precisão da deteção emocional ao facilitar a utilização de algoritmos especializados para cada sinal sonoro, visto que nem sempre a emoção transmitida pela lírica é a mesma transmitida pelo acompanhamento.

O Spleeter surgiu como uma biblioteca de código aberto destinada à separação de fontes sonoras, permitindo a investigadores e profissionais da área realizar esta tarefa de forma simples e eficiente. Disponibiliza modelos pré-treinados, capazes de realizar separações em diferentes granularidades (duas, quatro ou cinco fontes), o que simplifica o processo de extração de voz e instrumentos (Hennequin et al., 2020).

No sistema MERmaid, o Spleeter foi inicialmente integrado como solução de separação vocal, dada a sua simplicidade de implementação e os bons resultados obtidos. Contudo, surgiram limitações associadas à compatibilidade de versões e à ausência de manutenção ativa, motivando a transição para o Demucs, um sistema alternativo com ótimo desempenho e suporte técnico mais atualizado.

Embora o Spleeter tenha representado um avanço importante pela sua simplicidade e acessibilidade, a dependência de versões desatualizadas do TensorFlow e a falta de manutenção ativa acabam por limitar a utilização do mesmo em ambientes modernos. O Demucs surgiu assim como uma alternativa mais recente e robusta, desenvolvida pela equipa da Meta AI Research, baseada em PyTorch e concebida para oferecer uma separação precisa e natural de fontes sonoras.

A sua arquitetura combina camadas convolucionais e recorrentes, permitindo captar tanto características espectrais como temporais do áudio, o que resulta numa melhor preservação da qualidade vocal e instrumental (Défossez, 2021). Assim e graças à compatibilidade com ambientes Docker, o Demucs torna-se a solução mais adequada para o sistema MERmaid, substituindo o Spleeter.

4.2.3 Lyrics Retrival

O processo de *Lyrics Retrieval* (extração de letras musicais) procura isolar e transcrever o conteúdo textual de uma faixa de áudio, permitindo integrar a dimensão lírica em modelos de reconhecimento emocional. A relevância de cada dimensão, áudio ou letra, depende fortemente do estilo musical, sendo o áudio dominante em géneros de dança e a letra determinante em música de carácter poético (Malheiro et al., 2018).

Esta lírica pode ser obtida através de API externas ou modelos de reconhecimento de fala (*speech-to-text*). Ao recorrer a uma API externa, existe uma maior probabilidade de a lírica se encontrar correta; mas, ao mesmo tempo, existem restrições no número de pedidos diários possíveis ou erros devido à inconsistência dos formatos dos títulos.

Anteriormente, o projeto procurou introduzir este serviço ao utilizar a API *Genius*[3], mas enfrentou as restrições já referidas, o que limitou o serviço. Durante a iteração atual foi novamente testada esta técnica, ao fazer-se uso de outras API, mas repetiram-se os mesmos impasses.

Assim, foram testadas ferramentas de *Lyrics Retrieval* com modelos *speech-to-text*, neste caso o Whisper, desenvolvido pela equipa da OpenAI, obtendo-se resultados positivos. O Whisper é baseado em DL e treinado com elevados volumes multilingues e, através do uso de Docker e PyTorch, torna possível uma utilização *offline*.

A precisão dos resultados é influenciada pela qualidade do áudio utilizado, razão pela qual a separação de fontes sonoras anteriormente mencionada é relevante. Embora um processamento direto seja possível, quando aplicado a um áudio apenas lírico, os resultados tendem a apresentar um desempenho superior, visto que o ruído e a interferência instrumental são reduzidos.

4.2.4 Audio Segmentation

A segmentação de áudio consiste na divisão do ficheiro em partes mais pequenas e homogéneas. Assim, este processo assume um papel central na deteção de variação emocional na música, uma vez que os limites emocionais raramente coincidem com os limites estruturais, tornando-se mais natural, do ponto de vista do ouvinte, considerar a variação emocional como uma progressão de segmentos emocionalmente estáveis. Neste contexto, o desenvolvimento de métodos de segmentação mais eficazes torna-se crucial para que seja possível aplicar algoritmos de MER a problemas reais (Aljanaki et al., 2015).

No contexto MER, a segmentação permite uma análise granular, não considerando cada música como um todo estático, o que possibilita o estudo emocional evolutivo. Existem três principais métodos de segmentação: a baseada no silêncio(*Silence-based segmentation*), a temporal (*Time-based segmentation*) e a orien-

[3]https://docs.genius.com

tada ao ritmo ou às batidas musicais (*Beat-based segmentation*), como pode ser observado na Tabela 4.2.

Critério	**Segmentação temporal**	**Segmentação baseada no silêncio**	**Segmentação baseada em batidas**
Descrição	Divide o áudio em intervalos fixos (ex.: 5–10 segundos)	Identifica pausas naturais no sinal (baixa energia ou amplitude nula)	Utiliza deteção de batidas e variações rítmicas
Velocidade de Processamento	Alta	Média	Baixa
Utilização de Memória	Baixa	Moderada	Alta
Vantagens	Elevada consistência e baixo custo computacional	Simples, rápida e eficaz para deteção de separações naturais	Preserva a coerência musical e transições naturais
Desvantagens	Ignora variações musicais reais	Sensível a ruído e pausas falsas	Mais lenta e requer maior poder computacional
Adequação	Processamento rápido e previsível	Pré-processamento simples, considerando pausas naturais	Análise detalhada da estrutura rítmica / Géneros com forte batida

Tabela 4.2: Comparação entre métodos de segmentação áudio

No decorrer das iterações anteriores, o projeto MERmaid desenvolveu um microsserviço de segmentação baseado no método de separação temporal, através da utilização da biblioteca `fluent-ffmpeg`[4], que atualmente se encontra descontinuada.

Desta forma, foi utilizado o Librosa (McFee et al., 2015), biblioteca de análise de áudio, substituindo a ferramenta anterior e repondo o serviço. Esta possibilitou ainda escalar o serviço, permitindo a escolha entre cada um dos três tipos de segmentação.

4.2.5 Waveform Generation

A representação visual da forma de onda (*waveform*) permite observar graficamente a variação da amplitude do sinal ao longo do tempo. Esta visualização oferece uma perceção imediata da estrutura da música, identificando secções de maior intensidade, pausas ou transições dinâmicas.

No contexto do sistema MERmaid, esta onda contribui para a inspeção e validação do processamento áudio, como exemplificado na Figura 4.4, facilitando a

[4]https://www.npmjs.com/package/fluent-ffmpeg

interpretação dos resultados por investigadores e utilizadores.

Figura 4.4: Utilização da forma de onda como apoio interpretativo visual

Durante a primeira iteração do projeto, foi desenvolvido um serviço *WaveForm* em Node.js, com o objetivo de gerar representações visuais de forma de onda a partir de ficheiros áudio, permitindo uma melhor compreensão visual do conteúdo sonoro processado. A geração da imagem foi realizada através da biblioteca *generate-sound-waveform*, que utiliza o HTML5 Canvas para criar visualizações personalizadas.

O serviço permitia ainda ajustar parâmetros como a densidade da forma de onda, as cores, a espessura das linhas e a opacidade, possibilitando uma representação adaptada ao contexto visual pretendido. Esta abordagem facilitava tanto a inspeção manual dos ficheiros processados como a integração em interfaces gráficas, reforçando a transparência do processo de análise musical.

Tendo este sido desenvolvido na iteração inicial do projeto, não se encontra atualmente integrado no sistema. Será ainda interessante, se possível, integrar a ferramenta de geração de onda do projeto irmão, MERGE-APP, visto que foi desenvolvido recentemente e apresentou excelentes resultados por meio do uso da ferramenta `chart.js`.

4.2.6 Genre Classification

A classificação de género musical oferece ao utilizador uma caracterização estilística preliminar da faixa processada. Na versão atual do sistema, esta classificação é obtida através da API da Last.fm[5], que permite recuperar as etiquetas de género mais frequentes associadas ao artista ou à música específica.

[5] `https://www.last.fm/api`

O serviço opera de forma simples: o título e o artista são extraídos dos metadados do ficheiro, sendo posteriormente enviados para a API externa. São retornadas as etiquetas mais relevantes, das quais o sistema seleciona as cinco com maior pontuação, disponibilizando-as à interface.

Apesar da sua utilidade, esta abordagem apresenta limitações inerentes à dependência de uma base de dados colaborativa, sujeita a inconsistências, diferentes convenções de género e eventuais lacunas no catálogo. Por este motivo, encontra-se previsto o desenvolvimento de um classificador interno baseado em modelos de ML treinados especificamente para áudio musical. Tal evolução permitiria reduzir a dependência de serviços externos, melhorar a consistência dos resultados e incorporar múltiplas camadas de análise.

4.2.7 Emotion Classification

O módulo de classificação emocional complementa a análise musical ao atribuir etiquetas emocionais ao conteúdo sonoro. Na versão original do sistema, esta componente incluía a infraestrutura necessária para integrar um modelo de reconhecimento de emoção, mas não dispunha ainda de um modelo treinado, funcionando apenas como esqueleto funcional do microsserviço.

O objetivo deste módulo é identificar padrões emocionais expressos na voz, melodia, dinâmica e textura sonora, recorrendo a modelos que combinem técnicas de *audio feature extraction* com arquiteturas de aprendizagem profunda.

A evolução futura deste serviço prevê a integração de modelos especializados provenientes de trabalhos académicos e repositórios abertos, contribuindo para uma interpretação mais profunda e enriquecendo o ecossistema analítico do sistema.

4.2.8 Manager

O serviço *Manager* permite, na sua essência, controlar o fluxo de operações de um dado sistema. Desenvolvido em JavaScript, este desempenhava um papel de orquestração dentro da arquitetura original do sistema MERmaid, coordenando o fluxo de mensagens trocadas via RabbitMQ e assegurando a persistência de metadados através do MongoDB. Com o passar dos anos, o projeto evoluiu, deixando de lado a arquitetura original, e tornando obsoleta esta versão do serviço.

Na iteração atual, e com o desejo de integrar os diversos novos serviços mencionados, a presença de um serviço *Manager* tornou-se indispensável. Assim, foi proposta uma solução semelhante à visão inicial, respeitando o sistema atualizado e procurando simplicidade. Optou-se, então, por um desenvolvimento em Python, mantendo a comunicação assíncrona via RabbitMQ e a persistência de dados em MongoDB.

O MongoDB desempenhou um papel central no armazenamento dos dados de

processamento, que se distinguem pela sua natureza dinâmica e heterogénea. Cada entrada representa uma música e cada música apresenta: um estado de processamento, os serviços pelos quais passou e os resultados provenientes de cada um. O formato JSON[6] permite uma integração direta com os diversos serviços desenvolvidos e por desenvolver, visto oferecer um esquema flexível, ao contrário das bases de dados relacionais, adaptando-se às necessidades apresentadas. Assim, é possível adaptar rapidamente a estrutura à medida que os serviços são desenvolvidos, garantindo agilidade no desenvolvimento e fácil manutenção.

4.3 Limitações Identificadas

Devido à natureza extensa deste sistema, que envolve diversos microsserviços e múltiplos repositórios, a curva de aprendizagem torna-se mais acentuada. A manutenção de documentação clara, a nível de cada repositório, bem como global, poderia facilitar este processo, indicando em que estado foi deixado cada repositório, visto ser trabalhado ao longo dos anos por diversos alunos, de modo a facilitar o processo transitório. Uma boa documentação contribui ainda para a reprodutibilidade, essencial em ambientes de investigação.

A necessidade de respeitar normas, padrões e nomenclaturas deve ser realçada, visto permitir uma fluidez maior na navegação dos repositórios. Deve-se ter em conta que este não é um projeto comercial, mas sim de investigação, mantido por investigadores e alunos, e não por equipas dedicadas a tal tarefa, do que resulta uma evolução irregular do mesmo.

Este projeto, ao ter iterações anuais, sujeita-se ainda ao deterioramento de inúmeras ferramentas e dependências, como foi possível observar ao analisar os diversos microsserviços. Assim, procurou-se não cometer o mesmo erro, criando versões estáveis daqueles, recorrendo ao Docker e evitando o uso de inúmeras API externas, mantidas por terceiros e sem certezas de alterações de planos de utilização ou suporte futuro.

[6] JavaScript Object Notation

Esta página foi intencionalmente deixada em branco.

Capítulo 5

Implementação

Este capítulo abrange todo o processo de desenvolvimento, as ferramentas e as ações tomadas durante a iteração atual, considerando cada uma a finalidade, a longevidade e a identidade do projeto.

5.1 Manutenção Inicial do Sistema

No início do desenvolvimento, foram tomados cuidados normalmente aplicados em projetos de longa duração, focados na manutenção primária quando esta não se encontra automatizada.

5.1.1 Dependências

O desenvolvimento ocorreu inicialmente nos repositórios de interação com o utilizador: `mermaid-bk` e `mermaid-fe`, responsáveis, respetivamente, pela API e `front-end`. Procedeu-se à atualização de dependências de ambos, através da análise do comando `yarn outdated`[1], como demonstrado na Figura 5.1.

Esta manutenção periódica do sistema assegura a compatibilidade, desempenho e segurança. As atualizações podem ser classificadas segundo o seu impacto: as *patch* correspondem a correções menores de erros ou vulnerabilidades; as *minor* oferecem melhorias ou novas funcionalidades compatíveis com versões anteriores; as *intermidiate* representam alterações mais significativas, que em certos casos devem ser acompanhadas de pequenos ajustes no código; e as *major*, que implicam mudanças estruturais ou de interface, muitas vezes consideradas mudanças destrutivas, exigindo uma revisão e adaptação do sistema.

[1] `https://classic.yarnpkg.com/lang/en/docs/cli/outdated`

```
⊗ node ➜/workspaces/mermaid-fe (development) $ yarn outdated
  yarn outdated v1.22.22
  info Color legend :
   "<red>"    : Major Update backward-incompatible updates
   "<yellow>" : Minor Update backward-compatible features
   "<green>"  : Patch Update backward-compatible bug fixes
  Package                          Current Wanted  Latest  Package Type    URL
  @fortawesome/fontawesome-svg-core 6.5.1   6.7.2   6.7.2   dependencies    https://fontawesome.com
  @fortawesome/free-regular-svg-icons 6.5.1   6.7.2   6.7.2   dependencies    https://fontawesome.com
  @fortawesome/free-solid-svg-icons 6.5.1   6.7.2   6.7.2   dependencies    https://fontawesome.com
  @fortawesome/react-fontawesome    0.2.0   0.2.2   0.2.2   dependencies    https://github.com/FortAwesome/react-fontawesome
  @iconify/react                    4.1.1   4.1.1   6.0.0   dependencies    https://iconify.design/
  @testing-library/jest-dom         6.4.2   6.6.3   6.6.3   devDependencies https://github.com/testing-library/jest-dom#readme
  @testing-library/react            14.2.2  14.3.1  16.3.0  devDependencies https://github.com/testing-library/react-testing-library#readme
  @testing-library/user-event       14.5.2  14.6.1  14.6.1  devDependencies https://github.com/testing-library/user-event#readme
  axios                             1.6.8   1.9.0   1.9.0   dependencies    https://axios-http.com
  bootstrap                         5.3.3   5.3.6   5.3.6   dependencies    https://getbootstrap.com/
  chart.js                          4.4.4   4.4.9   4.4.9   dependencies    https://www.chartjs.org
  crypto-browserify                 3.12.0  3.12.1  3.12.1  dependencies    https://github.com/browserify/crypto-browserify
  dotenv                            16.4.5  16.5.0  16.5.0  dependencies    https://github.com/motdotla/dotenv#readme
  eslint                            8.57.0  8.57.1  9.27.0  devDependencies https://eslint.org
  eslint-config-prettier            9.1.0   9.1.0   10.1.5  devDependencies https://github.com/prettier/eslint-config-prettier#readme
  eslint-plugin-prettier            5.1.3   5.4.0   5.4.0   devDependencies https://github.com/prettier/eslint-plugin-prettier#readme
  eslint-plugin-react               7.34.1  7.37.5  7.37.5  devDependencies https://github.com/jsx-eslint/eslint-plugin-react
  i18next                           23.15.1 23.16.8 25.2.1  dependencies    https://www.i18next.com
  prettier                          3.2.5   3.5.3   3.5.3   devDependencies https://prettier.io
  react                             18.2.0  18.3.1  19.1.0  dependencies    https://react.dev/
  react-bootstrap                   2.10.2  2.10.10 2.10.10 dependencies    https://react-bootstrap.github.io/
  react-countup                     6.4.2   6.4.2   6.5.3   dependencies    https://react-countup.now.sh/
  react-dom                         18.2.0  18.3.1  19.1.0  dependencies    https://react.dev/
  react-i18next                     15.0.2  15.5.2  15.5.2  dependencies    https://github.com/i18next/react-i18next
  react-is                          18.2.0  18.3.1  19.1.0  dependencies    https://react.dev/
  react-parallax                    3.5.1   3.5.2   3.5.2   dependencies    https://github.com/rrutsche/react-parallax#readme
  react-router-dom                  6.23.1  6.30.1  7.6.1   dependencies    https://github.com/remix-run/react-router#readme
  socket.io-client                  4.7.5   4.8.1   4.8.1   dependencies    https://github.com/socketio/socket.io/tree/main/packages/socket.io-client#readme
  Done in 1.40s.
```

Figura 5.1: Resultado do comando `yarn outdated` para listar as versões das dependências presentes no repositório `mermaid-fe`

Foi então necessário proceder à atualização das dependências, de forma cuidada e estruturada, realizando sempre verificações após cada atualização, para manter o ambiente estável. Começando pelos *patch*, como previsto sem erros, procedeu-se às seguintes atualizações onde foram encontradas diversas incompatibilidades. Assim, foi decidido atualizar uma a uma, revertendo qualquer atualização prejudicial à estabilidade do sistema. Ao longo do desenvolvimento, e sempre que possível, foram ajustadas para versões estáveis, conforme a necessidade.

5.1.2 ESLint

Através do uso do ESLint, ferramenta de análise estática de código desenvolvida para o ecossistema JavaScript, foi possível identificar e corrigir automaticamente problemas de formatação e potenciais erros lógicos. A sua utilização foi integrada no sistema de manutenção de código, com o objetivo de garantir a uniformidade do estilo de escrita e a conformidade com as boas práticas de desenvolvimento.

A definição de regras personalizadas no ficheiro de configuração (`.eslintrc.json`), tornou possível alinhar o estilo de código entre os diferentes repositórios, facilitando a leitura e a revisão de código. O ESLint foi configurado para atuar em conjunto com o Prettier, assegurando a formatação automática dos ficheiros aquando dos *commits*.

```
1 # Analisar todos os ficheiros JavaScript do projeto
2 npx eslint .
3
4 # Corrigir automaticamente erros e formatação
```

```
5 npx eslint . --fix
```

Excerto 5.1: Execução do ESLint para análise e correção automática de código

Esta abordagem reduziu, significativamente, a ocorrência de erros triviais e de inconsistências sintáticas e promoveu um ambiente limpo e estável.

5.2 Microsserviços

Para além da organização funcional do sistema, os microsserviços encontram-se distribuídos em repositórios independentes, refletindo a sua modularidade e facilitando a sua manutenção, desenvolvimento e implantação. Esta separação permite atualizar ou substituir serviços de forma isolada, evitando impacto nos restantes componentes do *pipeline*. A Tabela 5.1 apresenta os principais repositórios associados ao sistema, bem como a finalidade de cada um.

Repositório	**Descrição**
`pipeline-manager-service`	Serviço responsável pela orquestração do *pipeline*, envio das mensagens e gestão da lógica de execução.
`youtube-downloader-service`	Serviço de obtenção de conteúdos a partir do YouTube, utilizando `yt-dlp` e `ffmpeg`.
`source-separation-service`	Serviço dedicado à separação de fontes musicais com Demucs, produzindo *stems* vocais e instrumentais.
`audio-segmentation-service`	Serviço de segmentação do áudio, suportando múltiplos métodos (tempo fixo, silêncio e batidas).
`lyrics-retrieval-service`	Serviço responsável pela transcrição de letras com Whisper, utilizando tanto o áudio original como *stems* vocais.
`wave2image`	Microsserviço que gera imagens de formas de onda a partir de ficheiros áudio através da biblioteca `generate-sound-waveform`.
`GenreFinder`	Serviço de classificação de género musical, responsável por extrair características musicais e devolver etiquetas estilísticas relevantes.
`emotion-classification-service`	Serviço de classificação de emoções na música, implementado com modelos de aprendizagem profunda.
`audio-processing-pipeline`	Repositório geral com documentação, scripts de suporte, artefactos partilhados e configuração agregada do sistema.
`Microservices-tryouts`	Espaço experimental para testes e prototipagem de componentes antes da sua integração no *pipeline* principal.

Tabela 5.1: Repositórios que compõem o ecossistema de microsserviços do pipeline.

Embora não essenciais, os repositórios `audio-processing-pipeline` e `Microservices-tryouts` permitiram um desenvolvimento local estável e a criação de um ambiente de testes e experimentação confortável.

A relação entre os microsserviços não é arbitrária: cada componente depende da produção de artefactos específicos gerados por serviços anteriores no *pipeline*. O diagrama da Figura 5.2 ilustra estas dependências diretas, evidenciando que o *YouTube Downloader* constitui o ponto de partida de todo o fluxo de processamento. A partir deste módulo, cada serviço consome o áudio descarregado ou os

resultados produzidos por etapas subsequentes, como a separação de fontes ou a transcrição. Esta representação visual clarifica a necessidade de uma ordem na execução e destaca as interações fundamentais dos serviços analisados nas secções seguintes.

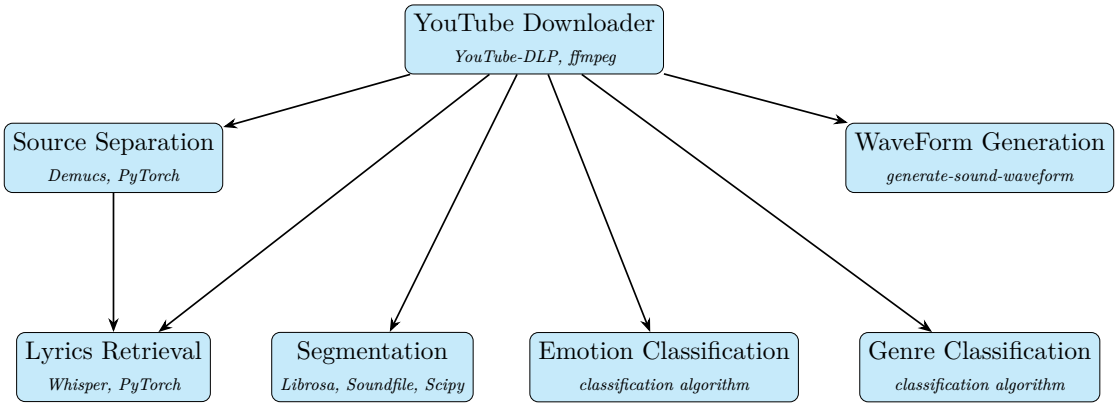

Figura 5.2: Diagrama de dependências diretas entre microsserviços no pipeline de processamento MERmaid v0.3

5.2.1 YouTube Downloader

O microsserviço de *download* do YouTube representa o ponto de entrada do *pipeline* de processamento de áudio, sendo responsável pela recolha e preparação dos ficheiros sonoros utilizados nas etapas subsequentes. Este componente recorre à ferramenta YouTube-DLP, uma derivação moderna do projeto YouTube-DL, que permite extrair faixas de áudio de diferentes plataformas de partilha de vídeo. De momento, apenas é utilizado o YouTube, mas o sistema está preparado para, se necessário no futuro, incluir novas plataformas.

Este opera de forma assíncrona através do RabbitMQ, recebendo o *link* ou o identificador do vídeo, previamente enviado pela API ao Manager. Desenvolvido em Python e executado num contentor Docker, o microsserviço utiliza o FFmpeg[2], para conversão e pós-processamento do áudio, e a biblioteca Pika[3], para comunicação com o *broker* de mensagens.

A arquitetura adotada é deliberadamente simples: não existe persistência em bases de dados, sendo os ficheiros áudio processados e colocados diretamente num volume partilhado, para poderem ser posteriormente consumidos por outros componentes do sistema. Esta escolha reduz o acoplamento[4] entre serviços e facilita a substituição ou melhoria futura do módulo de extração de áudio, se desejado.

A transição para o YouTube-DLP representou uma melhoria significativa em relação à solução anteriormente adotada, oferecendo maior estabilidade, mais funcionalidades e suporte a um amplo conjunto de plataformas. Esta atualização

[2]https://github.com/FFmpeg/FFmpeg
[3]https://pika.readthedocs.io/en/stable/intro.html
[4]*coupling*: grau de dependência entre componentes de um sistema

contribuiu para a fiabilidade do processo de ingestão, diminuindo falhas recorrentes observadas na versão antiga e garantindo maior robustez no início do fluxo de processamento.

O serviço apresenta ainda algumas limitações, nomeadamente a ausência de notificações automáticas sobre a conclusão do processo, sendo este controlo efetuado pelo *Manager*. Ainda assim, se isolado, apresenta um conjunto de *logs* diretos no terminal ou `stdout` do *container*, que permite ao utilizador conhecer o estado do processo a qualquer momento, bem como a localização final do ficheiro:

```
Connecting to RabbitMQ at rabbitmq:5672
YouTube Downloader listening on 'yt-dlp' queue
Received YouTube URL: https://www.youtube.com/watch?v=nVE1ziLuSNg
[youtube] Extracting metadata...
[info] Downloading format 18
[download] Destination: downloads/Sleep Token - Aqua Regia.mp4
[ExtractAudio] Destination: downloads/Sleep Token - Aqua Regia.mp3
Download completed: Sleep Token - Aqua Regia
File saved: downloads/Sleep Token - Aqua Regia.mp3
```

Excerto 5.2: Excerto de logs do serviço de download YouTube-DLP

Uso Isolado e Execução Local

A execução direta é feita através do script `downloader.py`, recebendo um URL e parâmetros opcionais de formato, qualidade e ativação de estratégias de recuperação. A utilização base foi mantida simples, permitindo gerar áudio em `.mp3` a 192 `kbps`. Contudo, visto que este é um trabalho de investigação, foram preparados formatos opcionais: WAV e FLAC, através do uso do FFmpeg, a integrar na página Web como opções para o utilizador no futuro, caso seja pertinente:

```
# Sintaxe base: URL obrigatório
python downloader.py <youtube_url> [format] [quality] [--fallback]

# Download simples em MP3 (192 kbps por omissão)
python downloader.py "https://www.youtube.com/watch?v=dQw4w9WgXcQ"

# Especificar formato WAV
python downloader.py "https://www.youtube.com/watch?v=dQw4w9WgXcQ" wav 320

# Exportar em FLAC
python downloader.py "https://www.youtube.com/watch?v=dQw4w9WgXcQ" flac

# Ativar estratégias alternativas de fallback
```

```
14 python downloader.py "https://www.youtube.com/watch?v=dQw4w9WgXcQ" mp3 192
       --fallback
```

Excerto 5.3: Exemplos de utilização do serviço localmente

Formato de Saída

As respostas estruturadas, usadas internamente, seguem um padrão simples: o sucesso, que inclui caminho, título, formato, qualidade e duração; e os erros, que incluem mensagem e, opcionalmente, sugestão de mitigação.

```
1 {
2   "success": true,
3   "file_path": "/downloads/Poppy - Computer Boy.mp3",
4   "title": "Poppy - Computer Boy",
5   "format": "mp3",
6   "quality": "192",      # kbps
7   "duration": 212,       # segundos
8   "file_size": 5123456   # bytes
9 }
```

Excerto 5.4: Resposta de sucesso do serviço de download

```
1 {
2   "success": false,
3   "error": "Video format not available. Try a different video or check if
       it's geo-blocked.",
4   "suggestion": "Some videos are restricted or use new formats. Try with a
       different video URL."
5 }
```

Excerto 5.5: Resposta de erro do serviço de download

A robustez deste microsserviço depende de manter o `yt-dlp` atualizado para acompanhar as mudanças frequentes na plataforma YouTube. O parâmetro de `fallback` é relevante perante mensagens como `"Requested format is not available"` ou falhas na extração. Os vídeos com duração superior a uma hora são rejeitados antes da fase de *download*, a fim de prevenir o consumo excessivo de recursos.

5.2.2 Source Separation

A separação de fontes constitui uma das etapas centrais do *pipeline* do sistema, permitindo decompor uma faixa de áudio nas suas componentes isoladas, normalmente designadas por *stems*. Este processo é particularmente relevante para ta-

refas de análise musical, extração de características, classificação e processamento avançado de áudio.

Tal como nos restantes módulos de processamento, a separação foi realizada de forma assíncrona através do RabbitMQ, com mensagens enviadas pelo Manager. Cada pedido indica o ficheiro áudio a processar e, opcionalmente, o modelo a utilizar. A implementação recorre ao Demucs, um dos modelos mais reconhecidos para separação de fontes musicais, que combina uma arquitetura baseada em redes neuronais profundas com elevada qualidade de reconstrução. A opção por este modelo garantiu resultados consistentes, permitindo isolar a voz, a bateria, o baixo e os restantes elementos instrumentais com elevada fidelidade.

O serviço foi desenvolvido em Python, refletindo a necessidade e a facilidade de integração com bibliotecas de processamento de áudio, amplamente utilizadas na comunidade científica. De forma semelhante ao microsserviço de *download*, a arquitetura é simples e direta: os stems produzidos são guardados diretamente num volume partilhado, sem persistência em base de dados, facilitando a sua utilização em qualquer etapa do *pipeline* de processamento. Esta abordagem reduz dependências e permite substituir ou atualizar o modelo utilizado no futuro, caso surjam alternativas mais adequadas ao contexto do projeto.

Antes de iniciar o processamento, o microsserviço permite a seleção de diferentes modelos fornecidos pelo Demucs, cada um com características distintas em termos de qualidade, velocidade de execução e otimização. Esta flexibilidade é particularmente útil em cenários de investigação, em que se podem comparar resultados, ajustar tempos de processamento ou equilibrar qualidade e desempenho, conforme necessário. Na Tabela 5.2 apresentam-se os principais modelos suportados pelo serviço.

Modelo	**Qualidade**	**Velocidade**	**Descrição**
`htdemucs`	Alta	Média	Modelo híbrido com transformadores (predefinido)
`mdx_extra`	Muito Alta	Lenta	Modelo com maior detalhe harmónico
`htdemucs_ft`	Superior	Lenta	Versão afinada para maior qualidade e fidelidade

Tabela 5.2: Modelos disponíveis para separação de fontes no Demucs

Atualmente, o serviço utiliza o modelo `htdemucs` como base, visto permitir um bom resultado a uma velocidade aceitável para o desenvolvimento contínuo, uma vez que a duração de cada processamento depende dos recursos disponíveis.

O microsserviço consome pedidos da fila `source-separation`, validando a existência do ficheiro de áudio e aplicando o modelo desejado ou o predefinido, quando não especificado. Cada processamento é executado sequencialmente, garantindo estabilidade, dado que o Demucs é computacionalmente exigente. A mensagem

que ativa o processamento segue o formato:

```
{
  "audio_file": "/app/input/ficheiro.mp3",
  "model": "htdemucs",
  "output_dir": "/app/separated"
}
```

Excerto 5.6: Formato da mensagem na fila source-separation

Os *logs* emitidos são informativos e permitem acompanhar o progresso do processo:

```
Connecting to RabbitMQ at rabbitmq:5672
Demucs Source Separator listening on 'source-separation' queue
Received separation request:
  File: /app/input/Radiohead - Karma Police.mp3
  Model: htdemucs
Starting source separation...
Source separation completed!
Created 4 stems: bass.mp3, drums.mp3, other.mp3, vocals.mp3
```

Excerto 5.7: Excerto de logs do serviço de separação

Estrutura de Saída

Os resultados seguem a estrutura padrão do Demucs, agrupando os *stems* num diretório organizado por modelo e nome do ficheiro processado, permitindo que os restantes componentes acedam facilmente aos resultados:

```
/app/separated/
        └── htdemucs/
            └── Radiohead - Karma Police/
                vocals.mp3
                drums.mp3
                bass.mp3
                other.mp3
```

Excerto 5.8: Exemplo da estrutura de saída para o serviço de separação de fontes

5.2.3 Lyrics Retrival

A extração de letras permitiu disponibilizar ao utilizador uma representação legível da componente vocal. Em vez de recorrer a bases de dados externas, frequentemente incompletas, inconsistentes ou sujeitas a restrições legais, o sistema opta

pela transcrição direta do áudio. Para tal, utiliza o modelo Whisper, uma solução robusta para reconhecimento automático de fala, capaz de lidar com ruído, variações de timbre, mistura instrumental e diferentes idiomas.

O microsserviço funciona de forma assíncrona através da fila `lyrics-transcription`. Cada pedido inclui o caminho do ficheiro a processar e, opcionalmente, o tamanho do modelo a utilizar:

```
1 {
2     "audio_file": "/app/input/Palaye Royale - Fever Dream.mp3",
3     "model_size": "medium"
4 }
```

Excerto 5.9: Exemplo de mensagem enviada para a fila lyrics-transcription

Esta escolha é particularmente relevante em cenários de investigação, porque permite alternar entre maior precisão (modelos `medium` ou `large`) e maior rapidez (modelos `tiny`, `base` ou `small`), consoante o contexto ou as restrições de recursos, como indicado na Tabela 5.3.

Modelo	**Velocidade**	**Descrição**
`tiny`	Muito Alta	Adequado para testes rápidos; precisão reduzida.
`base`	Alta	Compromisso equilibrado entre rapidez e qualidade (predefinido).
`small`	Média	Mais robusto em ambientes ruidosos; custo computacional moderado.
`medium`	Baixa	Melhor desempenho em passagens vocais difíceis ou misturas complexas.
`large`	Muito Baixa	Maior precisão; adequado para textos extensos ou línguas menos comuns.

Tabela 5.3: Modelos Whisper suportados pelo serviço de transcrição

O serviço valida o ficheiro, carrega o modelo solicitado e executa a transcrição, produzindo um ficheiro de texto num volume partilhado:

```
1 lyrics/
2     └── Djo - End Of Beginning_lyrics.txt
3     └── Djo - End Of Beginning_vocals_lyrics.txt
```

Excerto 5.10: Exemplo de ficheiro gerado pelo serviço de transcrição

O serviço gera artefactos textuais destinados às etapas posteriores do *pipeline* e, adicionalmente, persiste o conteúdo transcrito na base de dados do *Manager*. Esta integração permite à API realizar consultas diretas sem depender do sistema de ficheiros e reforça a consistência entre os diferentes componentes da plataforma.

Um aspeto relevante deste serviço é a sua compatibilidade com a separação de fontes. Quando o ficheiro de entrada corresponde a um *stem* vocal, o sistema adapta a nomenclatura do ficheiro transcrito, permitindo distinguir explicitamente as letras extraídas da mistura completa das letras geradas a partir da voz isolada. Esta característica potencia análises comparativas, embora ainda não tenham sido implementadas.

Durante uma utilização normal do serviço, é transcrito o componente lírico do áudio original, seguido da transcrição do correspondente ficheiro vocal, completando o ciclo de transcrições pretendido. Os registos produzidos pelo microsserviço permitem acompanhar as várias etapas do processo, desde a ligação ao *broker* até à criação dos ficheiros finais:

```
Connecting to RabbitMQ at rabbitmq:5672
Whisper Lyrics Transcriber listening on 'lyrics-transcription' queue
Waiting for transcription requests...

Received transcription request:
File: /app/input/Djo - End Of Beginning.mp3
Model: medium
Loading Whisper model: medium
Whisper model loaded successfully
Starting transcription...
Transcribing: Djo - End Of Beginning.mp3
Transcription completed successfully!
Output: lyrics/Djo - End Of Beginning_lyrics.txt
Language: en
Text length: 878 characters

Received transcription request:
File: /app/separated/htdemucs/Djo - End Of Beginning/vocals.mp3
Model: medium
Loading Whisper model: medium
Whisper model loaded successfully
Starting transcription...
Transcribing: vocals.mp3
Transcription completed successfully!
Output: lyrics/Djo - End Of Beginning_vocals_lyrics.txt
Language: en
Text length: 874 characters
```

Excerto 5.11: Excerto de logs do serviço de transcrição

Como é possível observar no excerto 5.11, quando a mesma música é transcrita com ou sem acompanhamento, é possível obter resultados diferentes, conforme confirmado pelo número de caracteres transcritos. Não foi construido um serviço para

comparação destes resultados, mas, após observação ao longo do desenvolvimento, confirmou-se uma discrepância em torno de 5-10 % no número de caracteres, bem como uma diferença de 10-15 % entre os dois resultados produzidos.

Análise de Resultados

A avaliação da qualidade das transcrições foi realizada através da métrica Word Error Rate (Taxa de Erro de Palavras) (WER), amplamente utilizada na literatura de reconhecimento automático de fala. Embora o sistema não implemente um cálculo automático desta métrica, a análise manual realizada durante o desenvolvimento revelou tendências claras no comportamento dos diferentes modelos.

Tal como apresentado na Tabela 5.4, modelos menores enfrentam perdas significativas em ambientes musicais complexos, onde a presença de instrumentação densa, reverberação e sobreposições vocais introduz desafios adicionais que não se verificam em cenários de fala isolada.

Modelo	**WER Obtido (%)**	**Notas**
`tiny`	18-25 %	Elevada rapidez ao custo da precisão; erros acentuados em vocais com ruído ou mistura forte.
`base`	14–20 %	Desempenho equilibrado; adequado para transcrição geral sem requisitos de alta fidelidade.
`small`	10–15 %	Melhoria consistente em ambientes ruidosos; bom compromisso entre precisão e recursos.
`medium`	8–10 %	Significativamente mais robusto em vocais difíceis e dicção irregular.
`large`	3–6 %	Melhor precisão global; mais estável em música complexa, misturas densas ou línguas menos comuns.

Tabela 5.4: Valores obtidos de taxa de erro de palavras para diferentes modelos Whisper.

Durante o desenvolvimento, verificou-se que o modelo `medium` oferece um equilíbrio adequado entre custo computacional e precisão, sendo capaz de capturar a grande maioria das palavras, mesmo em excertos difíceis, tornando-se, por isso, o modelo predefinido utilizado pelo serviço. A utilização de modelos maiores, como o `large`, revelou ganhos adicionais na fidelidade da transcrição, mas a exigência computacional tornou-se inadequada para o processamento contínuo.

A discrepância observada entre as transcrições do áudio misturado e do *stem* vocal reforça a relevância de aplicar separação de fontes antes da transcrição, uma vez que tende a melhorar a inteligibilidade da voz e a reduzir erros recorrentes:

```
# saturdays at your place - tarot cards (Excerto - Original)
We're getting off to a rough start
Saw you spill your drink from around the corner
You might think I'm awkward
I'm always checking the time

# saturdays at your place - tarot cards (Excerto - Lyrics)
We're getting off to a rough start
Size will your drinks are around the corner
You might think of my word
I'm always checking the time

# saturdays at your place - tarot cards (Excerto - Lyrics - Vocals)
We're getting off to a rough start
So you spill your drink from around the corner
You might think I'm awkward
I'm always checking the time.
```

Excerto 5.12: Discrepância existente na transcrição do áudio separado

Embora este comportamento seja o mais comum, também se observaram casos inversos, nos quais o áudio original produziu resultados superiores aos obtidos após a separação de fontes:

```
# Skott - Mermaid (Excerto - Original)
I'll be your mermaid
Caught on your rock
Coming for your aid
Isn't it odd?
Isn't it silly
Now that you know?
Someone this slippery
Can't let you go

# Skott - Mermaid (Excerto - Lyrics)
I'll be your mermaid
Caught on your rock
Coming for your aid
Isn't it odd?
Isn't it silly
Now that you know?
Sound on the slippery
Can't let you go
```

```
# Skott - Mermaid (Excerto - Lyrics - Vocals)
I'll be your mermaid
God on your rock
Coming for your e
Isn't it odd?
Is it silly now that you know
Sound on the slippery
Can't let you go
```

Excerto 5.13: Discrepância existente na transcrição do áudio não separado

Estes resultados contrastantes evidenciam a importância de testar múltiplas abordagens num contexto de investigação, em vez de assumir que um único método é universalmente superior. Cada estratégia apresenta vantagens e limitações, sendo ambas relevantes para o desenvolvimento contínuo do projeto.

Desenvolvimento Futuro

Uma evolução natural deste microsserviço seria a integração de um módulo automático de cálculo da WER, permitindo comparar de forma sistemática as transcrições produzidas pelos diferentes modelos do Whisper ou por versões distintas do serviço. A automatização desta métrica possibilitaria quantificar melhorias, identificar regressões, avaliar o impacto de novas técnicas de pré-processamento e validar alterações ao *pipeline* de forma objetiva. A existência deste mecanismo contribuiria, assim, para uma monitorização contínua da qualidade do sistema e para uma avaliação mais rigorosa do seu desempenho em cenários reais.

Para além da análise da qualidade da transcrição, torna-se igualmente pertinente explorar técnicas de análise textual aplicadas especificamente a letras musicais. A integração futura de modelos de análise de sentimento, ajustados ao conteúdo musical, bem como métodos de extração semântica e identificação de temas ou emoções predominantes, enriquece o valor interpretativo do sistema.

5.2.4 Audio Segmentation

Tal como os restantes microsserviços, a segmentação de áudio recebe do *Manager* o ficheiro a processar e o método pretendido. A implementação foi realizada em Python, recorrendo ao Librosa para análise e processamento de áudio e ao Soundfile para a escrita dos segmentos resultantes. Tal como os restantes serviços, os ficheiros são guardados diretamente num volume partilhado, não recorrendo a uma base de dados, pelas mesmas razões já referidas.

Foram implementados três métodos distintos de segmentação, cobrindo necessidades diferentes: segmentação temporal, segmentação baseada em silêncio e segmentação baseada em batidas. Esta diversidade permite adaptar o comportamento do serviço à natureza do conteúdo processado ou rumo de investigação.

Integração e Fluxo de Processamento

A interação com o resto da plataforma é realizada através do RabbitMQ na fila `audio-segmentation`. Cada mensagem especifica o ficheiro a segmentar, o método pretendido e os parâmetros associados. O formato das mensagens permite ajustar o comportamento do serviço de forma granular:

```
{
  "audio_file": "/app/input/ficheiro.wav",
  "method": "time",
  "segment_length": 30,
  "overlap": 5,
  "threshold": -40,
  "min_silence": 0.5,
  "min_segment": 2.0,
  "segments_per_beat": 4
}
```

Excerto 5.14: Formato da mensagem na fila audio-segmentation

Para ilustrar a forma como cada método é configurado, apresentam-se exemplos de mensagens enviadas para a fila `audio-segmentation`. Cada exemplo destaca apenas os parâmetros relevantes para o respetivo método:

```
{
  "audio_file": "/app/input/Bad Guy.mp3",
  "method": "time",
  "segment_length": 30,
  "overlap": 5
}
```

Excerto 5.15: Exemplo de uma mensagem para segmentação por tempo

```
{
  "audio_file": "/app/input/Caramel.mp3",
  "method": "silence",
  "threshold": -40,
  "min_silence": 0.5,
  "min_segment": 2.0
}
```

Excerto 5.16: Exemplo de uma mensagem para segmentação por silêncio

```
{
  "audio_file": "/app/input/505.mp3",
  "method": "beats",
  "segments_per_beat": 4
```

```
5 }
```

Excerto 5.17: Exemplo de uma mensagem para segmentação por batidas

Os registos emitidos durante a execução permitem acompanhar o processo, facilitando a monitorização:

```
1 Connecting to RabbitMQ at rabbitmq:5672
2 Audio Segmenter listening on 'audio-segmentation' queue
3 Waiting for segmentation requests...
4 Received segmentation request:
5 File: /app/downloads/Three Days Grace - Dominate Lyric Video.mp3
6 Method: time
7 Length: 30s, Overlap: 0.0s
8 Segmentation completed!
9 Total segments: 6
10 Duration: 195.19s
```

Excerto 5.18: Excerto de logs do serviço de segmentação

Uso Isolado e Execução Local

Para efeitos de validação, o serviço pode ser executado fora do *pipeline*. O *script* `segmenter.py` disponibiliza diretamente os três métodos implementados, permitindo testar parâmetros, analisar resultados e verificar tempos de execução:

```
1 python segmenter.py "Bad Guy.mp3" time 30 5
2 python segmenter.py "Caramel.mp3" silence -40 0.5 2.0
3 python segmenter.py "505.mp3" beats 4
```

Excerto 5.19: Exemplos de utilização do serviço de segmentação no terminal

Estrutura de Saída

Os segmentos são organizados num diretório próprio por música, seguindo convenções de nome que identificam claramente o método utilizado e de que segmento se trata. Esta organização simplifica o consumo posterior pelos restantes serviços do sistema.

```
1 segments/
2     └── Joji - Glimpse of Us/
3         Joji - Glimpse of Us_segment_000.wav
4         Joji - Glimpse of Us_silence_seg_000.wav
5         Joji - Glimpse of Us_beat_seg_000.wav
```

Excerto 5.20: Estrutura de saída produzida

5.2.5 Emotion Classification

A implementação atual do microsserviço de classificação emocional foi concebida como uma solução transitória, destinada a manter a coerência estrutural do *pipeline* e validar a integração entre os serviços. Uma vez que o modelo de classificação emocional previsto para esta versão, proveniente de um projeto irmão, ainda não se encontrava disponível no momento do desenvolvimento deste projeto, optou-se por introduzir um mecanismo simplificado que permite ao sistema produzir uma etiqueta emocional válida para cada áudio processado, garantindo a continuidade funcional do fluxo.

Para o efeito, o serviço gera uma categoria emocional através da seleção aleatória a partir de um conjunto fixo de cinco classes: `Happy`, `Sad`, `Tense`, `Calm` e `Mixed`. Esta abordagem, embora não represente o respetivo conteúdo sonoro, cumpre dois propósitos essenciais: possibilita a validação de toda a cadeia de comunicação entre a API, o *Manager* e a aplicação Web; e assegura que os campos e estruturas de dados associados à classificação emocional sejam corretamente preenchidos, armazenados em MongoDB e integrados na interface.

A decisão de incluir um *placeholder*, em vez de remover temporariamente o módulo, foi particularmente relevante para preservar a compatibilidade com o design global da plataforma. A existência do serviço permite testar as dependências, *endpoints*, rotas de resultados e visualizações finais, evitando alterações estruturais significativas aquando da eventual integração do modelo definitivo. Além disso, a inclusão de uma etiqueta emocional, ainda que simbólica, apoia o desenvolvimento incremental da camada de visualização, permitindo, desde já, testar formatos de exibição, métricas associadas e interações com outros componentes analíticos.

Numa fase futura, o módulo será substituído por um classificador real baseado em técnicas de *audio feature extraction* e aprendizagem profunda. A infraestrutura atual foi desenhada para facilitar essa substituição, exigindo apenas a troca do mecanismo de inferência, mantendo intactos os formatos de entrada, saída e persistência. Desta forma, o desenvolvimento realizado nesta versão estabelece as bases para a integração direta de modelos avançados provenientes do projeto colaborativo que se encontra em desenvolvimento.

5.3 Manager

O *Manager* constitui o núcleo de coordenação do sistema, responsável pelo fluxo de processamento entre os vários microsserviços, pela gestão do estado global de cada pedido e por persistir toda a informação relevante sobre as etapas executadas.

Sempre que um *link* é submetido para processamento, o Manager cria um registo inicial em MongoDB e desencadeia o *pipeline* completo, garantindo que cada fase é executada na ordem apropriada. Esta sequência é tratada como um grafo linear de dependências, como ilustrado na Figura 5.3, onde cada etapa espera que a anterior seja concluída com sucesso.

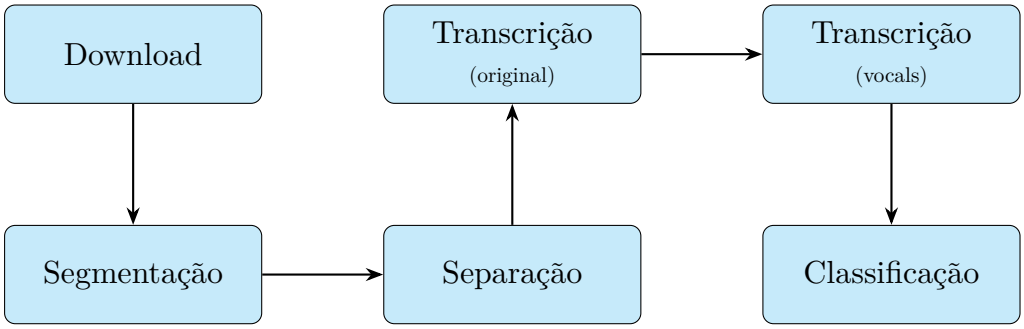

Figura 5.3: Fluxo de processamento

A comunicação entre os componentes é inteiramente mediada através do RabbitMQ, assegurando baixo acoplamento e elevada resiliência. A Tabela 5.5 sintetiza as principais filas utilizadas no sistema e o papel do Manager em cada uma delas.

Fila RabbitMQ	**Função**
`manager-requests`	Fila de entrada de pedidos de processamento.
`yt-dlp`	Pedido de *download* da faixa áudio.
`audio-segmentation`	Pedido de segmentação do ficheiro descarregado.
`source-separation`	Pedido de separação em *stems*.
`lyrics-transcription`	Pedido de transcrição (completa e voz isolada).

Tabela 5.5: Filas RabbitMQ utilizadas e respetiva função.

5.3.1 Integração e Fluxo de Processamento

A entrada de pedidos ocorre através de mensagens publicadas na fila `manager-requests`. O *manager* consome essas mensagens, extrai o `video_id` do URL e cria um documento inicial na coleção `videos` na base de dados `audio_pipeline`. Cada documento contém o estado global do vídeo e o estado individual de processamento de cada serviço, inicialmente definidos como `pending`.

O envio de pedidos ao *manager* ocorre na fila `manager-requests`. A mensagem é reduzida ao essencial: o *link* é enviado em texto simples, mantendo o consumidor responsável pela extração do identificador e pela criação da estrutura de dados correspondente:

```
1 # Exemplo da utilização local
2 python send_request.py "https://www.youtube.com/watch?v=Xe7TTVOAPIk"
3
4 # Saída típica:
5 Successfully sent to manager:
6 URL: https://www.youtube.com/watch?v=Xe7TTVOAPIk
7 Queue: manager-requests
```

Excerto 5.21: Exemplo de envio de pedido ao Manager

5.3.2 Modelo de Dados

Cada vídeo processado é representado por um documento único na coleção `videos`. O *manager* cria índices em `video_id`, `created_at` e `status`, permitindo consultas eficientes e escaláveis, permitindo compilar as informações mais importantes para cada áudio individual:


```
{
  "video_id": "nVE1ziLuSNg",
  "url": "https://www.youtube.com/watch?v=nVE1ziLuSNg",
  "title": "Sleep Token - Aqua Regia",
  "status": "completed",
  "created_at": "2025-10-29T16:49:21.346000",
  "updated_at": "2025-11-28T17:11:30.734000",
  "stages": {
    "download": {
      "status": "completed",
      "started_at": "2025-11-28T17:06:09.423000",
      "completed_at": "2025-11-28T17:06:34.366000",
      "output_file": "/app/downloads/Sleep Token - Aqua Regia.mp3",
      "error": null
    },
    "segmentation": {
      "status": "completed",
      "started_at": "2025-11-28T17:06:34.367000",
      "completed_at": "2025-11-28T17:06:46.298000",
      "output_folder": "/app/segments/Sleep Token - Aqua Regia",
      "segment_count": 7,
      "error": null
    },
    "separation": {
      "status": "completed",
      "started_at": "2025-11-28T17:06:46.299000",
      "completed_at": "2025-11-28T17:06:56.306000",
      "output_folder": "/app/separated/htdemucs/Sleep Token - Aqua Regia",
      "stems": [
        "bass",
        "vocals",
        "other",
        "drums"
      ],
      "error": null
    },
```

```
37    "transcription_full": {
38      "status": "completed",
39      "started_at": "2025-11-28T17:06:56.307000",
40      "completed_at": "2025-11-28T17:09:41.006000",
41      "output_file": "/app/lyrics/Sleep Token - Aqua Regia_lyrics.txt",
42      "word_count": 237,
43      "error": null,
44      "lyrics": "Well my love is an animal call Cutting through the
    darkness, bouncing off the walls Between teeth on a broken jaw
    Following a blood trail frothing out the mower These days I'm a
    circuit board Integrated hardware you cannot afford The perfect start
    to a perfect war Putting down the roses, picking up the sod A quote,
    read up A quote, read up Well Mark Paz is a holy book A call from
    Olympus, ringing off the hook Between the pain and the way you look I'
    m stuck in a time where the mountains should These days I'm a picture
    frame Screaming out the sunshine, singing in the rain Sugar on the
    blood cells, carbon on the brain Olive Eden's spices, running through
    my veins A quote, read up Oxytocin running in the evil Silicon, all
    rooms Subatomic interactions if it's all good Gold rush, acid flux
    Saturate me, I can't get enough Cold blood, hot blood Run it to your
    heart, what you're thinking of Oh and I am done Dancing to a lullabies
     No wonder my ears are still ringing And I am done Fighting off change
     No wonder my arms are still swinging A quote, read up Oxytocin
    running in the evil Silicon, all rooms Subatomic interactions if it's
    all good Gold rush, acid flux Saturate me, I can't get enough Cold
    blood, hot blood Run it to your heart, what you're thinking of"
45    },
46    "transcription_vocals": {
47      "status": "completed",
48      "started_at": "2025-11-28T17:09:41.007000",
49      "completed_at": "2025-11-28T17:11:30.732000",
50      "output_file": "/app/lyrics/Sleep Token - Aqua Regia_vocals_lyrics.
    txt",
51      "word_count": 229,
52      "error": null,
53      "lyrics": "Well, my love is an animal call Cutting through the
    darkness, bouncing off the walls Between teeth on a broken jaw
    Following a blood trail frothing out the moor And these days I'm a
    circuit board Integrated hardware you cannot afford The perfect start
    to a perfect war Putting down the roses, picking up the sod Aqa, Reja
    Aqa, Reja Well, Marpass is a holy book A call from Olympus, ringing
    off the hook Between the pain and the way you look I'm stuck in a time
```

```
     where the mountains shook And these days I'm a picture frame
    Screaming out the sunshine, singing in the rain Sugar on the blood
    cells, carbon on the brain Olive Eden's spices running through my
    veins Aqa, Reja Oxytocin running in the knee Second, ball groans
    Subatomic interactions if it's all good Gold rush, acid flux Saturate
    me, I can't get enough Cold blood, hot blood Run it to your heart when
     you're thinking of Oh, and I am done Dancing to Olympus No wonder my
    ears are still ringing And I am done Fighting off change No wonder my
    arms are still swinging Aqa, Reja Oxytocin running in the knee Second,
     ball groans Subatomic interactions if it's all good Gold rush, acid
    flux Saturate me, I can't get enough Cold blood, hot blood Run it to
    your heart when you're thinking of"
    },
    "emotion_classification": {
      "status": "completed",
      "started_at": "2025-11-28T17:11:30.733000",
      "completed_at": "2025-11-28T17:11:30.733000",
      "emotion": "Happy",
      "error": null
    }
  },
  "metadata": {
    "duration_seconds": 236.1,
    "file_size_mb": 5.41,
    "processing_time_seconds": 321.3
  }
}
```


Excerto 5.22: Documento exemplo para o vídeo *Sleep Token - Aqua Regia*

A estrutura do documento apresentado reflete a lógica interna do sistema de processamento e a função central do *Manager* enquanto agregador de resultados. Cada vídeo corresponde a um único documento, funcionando como ponto de convergência de todas as etapas do *pipeline*. Esta abordagem elimina a necessidade de múltiplas coleções interdependentes, reduz complexidade e facilita a obtenção de um retrato completo do estado de processamento.

Os campos iniciais: `video_id`, `status`, `created_at` e `updated_at`; permitem identificar rapidamente a entrada e acompanhar a evolução temporal do processamento. De seguida, o bloco `stages` organiza a informação de forma modular, atribuindo a cada etapa um subdocumento próprio, que contém não só o estado, `pending`, `processing`, `completed` ou `failed`, mas também os artefactos produzidos, registos temporais e erros associados. Esta granularidade é essencial para a deteção de anomalias, a análise do comportamento do *pipeline* e a execução seletiva de etapas, evitando repetir o processamento completo.

A inclusão das letras completas nos subdocumentos de transcrição, bem como os resultados da classificação emocional, demonstra a capacidade do modelo de absorver novos serviços sem alterar a estrutura existente. Por fim, a secção `metadata` agrega informação transversal, como a duração do áudio, o tempo total de processamento ou o tamanho dos ficheiros, para suportar futuramente análises de desempenho, otimização ou visualizações na aplicação Web.

A indexação explícita dos campos `video_id`, `status` e `created_at` garante que operações frequentes, como listagens filtradas, consultas individuais ou estatísticas globais, permaneçam eficientes, mesmo com o crescimento da base de dados. Em conjunto, o modelo de dados do *Manager* constitui uma fundação robusta para a evolução da plataforma, suficientemente estruturada para permitir extensões, mas flexível para acomodar novos serviços e resultados.

5.3.3 Consulta de Estado e Estatísticas

Foram ainda criados *scripts* como o `query_status.py` que disponibiliza três funcionalidades principais de consulta: a de todos os vídeos, a de um vídeo em específico e a das estatísticas gerais:

```
# Listar vídeos (filtro opcional)
python query_status.py list
python query_status.py list completed

# Consultar um vídeo específico
python query_status.py get nVE1ziLuSNg

# Consultar o json de um vídeo específico
python query_status.py json nVE1ziLuSNg

# Estatísticas agregadas do sistema
python query_status.py stats
```

Excerto 5.23: Consultas suportadas

É ainda possível observar o completo fluxo de processamento através dos *logs* criados pelo serviço, presentes no Apêndice A.

5.4 API

Durante o trabalho realizado, as alterações introduzidas foram pontuais e focadas, sobretudo, na integração com o *pipeline* de processamento e na melhoria da segurança do sistema.

A principal modificação consistiu na adição de um novo *endpoint*, responsável por encaminhar pedidos de processamento de áudio diretamente para o *manager*, abstraindo a complexidade das filas do RabbitMQ do lado do cliente. Este *endpoint* recebe o URL fornecido pelo utilizador, valida-o e publica o pedido na fila apropriada. A integração permitiu unificar o fluxo de operação, garantindo que todo o processamento passa exclusivamente pelo *manager*, evitando acessos diretos dos utilizadores aos microsserviços.

Outra alteração relevante foi a migração da `YouTube API key` da interface Web para a API. Anteriormente, a chave encontrava-se embutida no código da interface Web, o que apresentava riscos de exposição e uso indevido. Esta alteração reforçou a segurança, pois permite controlar o acesso e monitorizar potenciais abusos, além de alinhar o projeto com boas práticas de desenvolvimento seguro. Foi ainda melhorada a pesquisa associada ao pedido, apresentando apenas vídeos de música, limitação ausente nas versões anteriores da aplicação.

Adicionalmente, foi corrigido o sistema de WebSockets responsável por transmitir, em tempo real, o estado de processamento ao cliente. A implementação original encontrava-se inoperacional, impedindo a atualização dinâmica da interface Web. Após a correção, o canal de comunicação passou a refletir corretamente o avanço do *pipeline* para cada pedido. Para garantir resiliência, foi também introduzido um mecanismo complementar de *polling*, utilizado como estratégia de tolerância a falhas: caso uma mensagem WebSocket seja perdida ou o cliente restabeleça a ligação, a aplicação Web pode recorrer a este método para recuperar o estado atual do processamento e evitar inconsistências na apresentação.

Inicialmente, foram ponderadas mudanças estruturais adicionais à API, embora descartadas. Os restantes *endpoints* mantiveram-se funcionais e suficientes para as necessidades da aplicação Web, razão pela qual não se justificou qualquer reestruturação profunda nesta fase do desenvolvimento.

5.5 Aplicação Web

A aplicação Web manteve-se, tal como a API, com um conjunto reduzido de alterações estruturais. O objetivo desta fase do trabalho não foi redesenhar a interface, mas integrá-la de forma consistente com o novo fluxo de processamento.

A principal modificação introduzida consistiu na atualização do formulário de submissão: em vez de comunicar diretamente com o serviço de *download*, a aplicação passa agora a enviar o pedido para a API, que, por sua vez, encaminha o URL para o *manager*. Esta alteração unificou o fluxo, eliminando caminhos paralelos e reduzindo a complexidade do sistema.

Foram também efetuadas duas pequenas melhorias relacionadas com a validação de URL: o tratamento de erros enviados pela API; a apresentação do estado do pedido, ainda que de forma simplificada. A implementação de um painel de progresso tornou-se possível após a correção do mecanismo de WebSockets, permi-

tindo apresentar, na interface, a percentagem de avanço de cada música presente na fila de processamento. Assim, esta funcionalidade deixou de depender de *endpoints* adicionais no *manager*, passando a ser suportada de forma integrada pela comunicação em tempo real.

Por fim, foi adicionada uma nova secção na página individual de cada música, que permite ao utilizador aceder aos artefactos gerados durante a análise. Esta secção inclui os segmentos de áudio, correspondentes às diferentes secções temporais identificadas na música e apresentados em formato WAV através de leitores de áudio HTML5 integrados na interface; inclui também como os *stems* resultantes da separação de fontes sonoras, disponibilizados em formato MP3, bem como faixas individuais para vocais, baixo, bateria e outros instrumentos, igualmente reproduzíveis diretamente na aplicação. São ainda disponibilizadas as transcrições textuais das letras obtidas através do serviço de reconhecimento de fala, em duas versões distintas: a transcrição completa do áudio original e a transcrição apenas da faixa vocal isolada.

A implementação desta funcionalidade exigiu a criação de novos *endpoints* na API para obtenção dos metadados de cada artefacto, bem como a configuração do servidor para disponibilizar os ficheiros de áudio, permitindo que estes possam ser reproduzidos diretamente na aplicação Web. Na interface, recorreu-se a componentes Accordion do Bootstrap para organizar a informação de forma colapsável, melhorando a experiência de navegação sem sobrecarregar visualmente a página.

Esta adição representa um passo importante na transparência do sistema, pois permite aos utilizadores não apenas visualizar os resultados da classificação emocional, e também explorar os dados intermédios que fundamentam essa análise.

5.5.1 Interface da Aplicação Web

Para complementar a descrição funcional da aplicação Web, apresentam-se alguns excertos da interface que ilustram o fluxo de utilização, desde a submissão do URL até à consulta dos resultados processados. As imagens não têm como objetivo detalhar o design visual, mas sim documentar o comportamento atual da interface e a sua integração com a API e o *manager*.

A página inicial apresenta um campo simples para a introdução do URL, como demonstrado na Figura 5.4. Este campo funciona com um breve atraso, de forma a evitar um uso indesejado de chamadas à API do YouTube antes de o utilizador terminar a escrita.

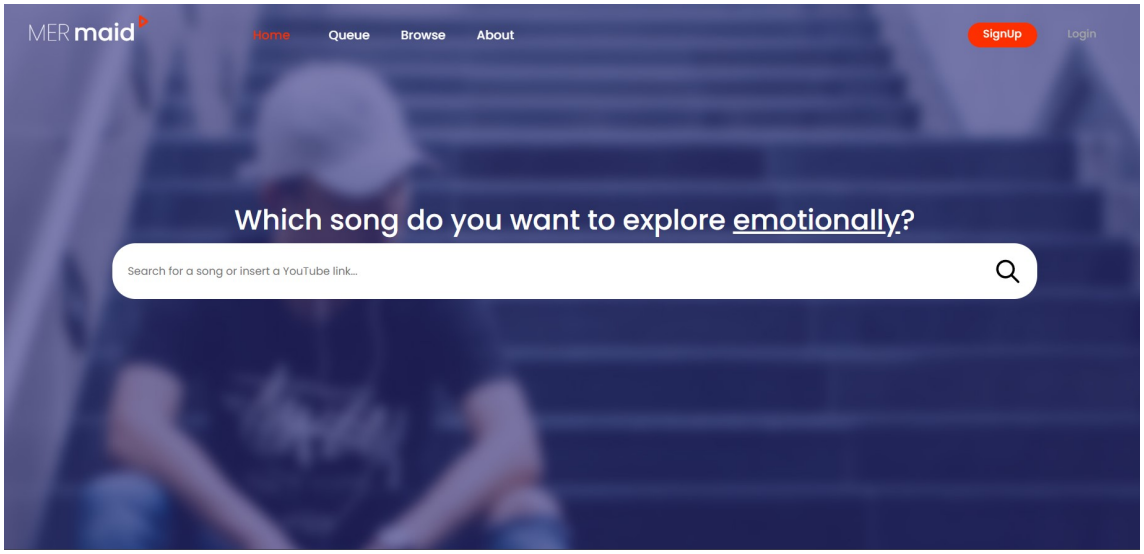

Figura 5.4: Página inicial da aplicação Web

Após esta pesquisa, é possível observar até quatro músicas a classificar e um botão `Classify` para iniciar o processamento independente de cada música (Figura 5.5).

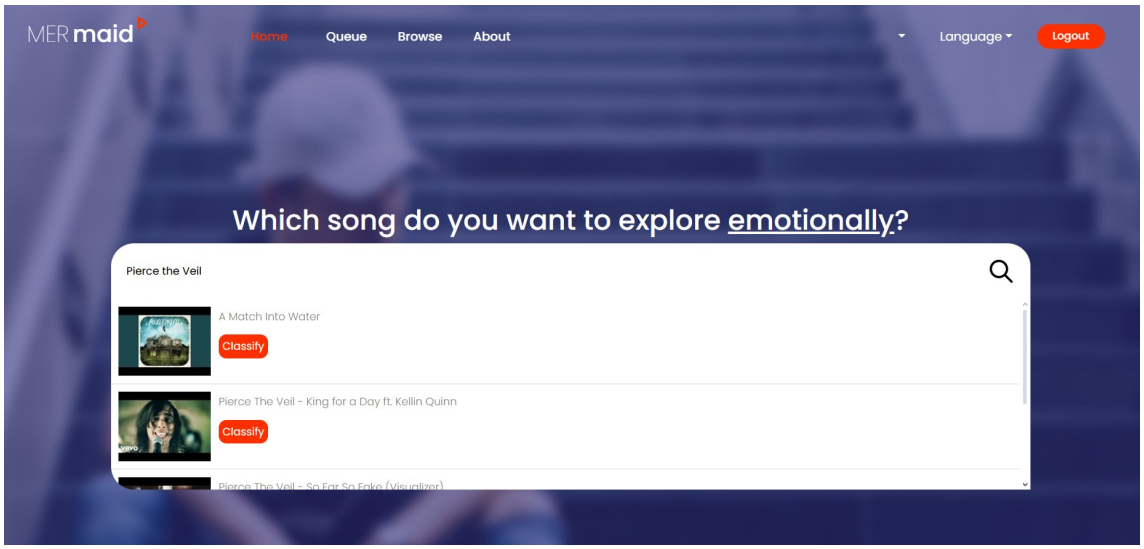

Figura 5.5: Utilização da barra de pesquisa e processamento

É ainda possível observar uma secção que indica o total de minutos e o número de vídeos analisados, como demonstrado no Apêndice B.

A página `Queue` reflete o estado atual do sistema. Após ser realizado um pedido de classificação, este é enviado à API e, consequentemente, ao *Manager*. Sempre que existam músicas ativas no *pipeline*, esta secção apresenta a lista de pedidos em curso, acompanhada da respetiva percentagem de progresso (Figura 5.6). Isto permite ao utilizador perceber rapidamente se o pedido já se encontra em processamento ou em espera, visto que apenas uma música pode ser processada de cada vez.

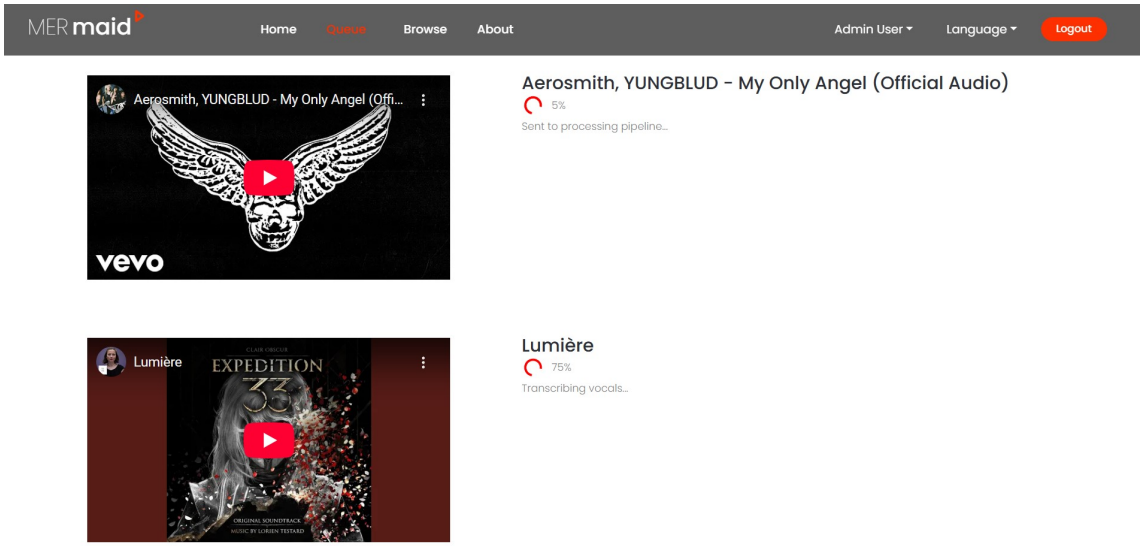

Figura 5.6: Página da fila de processamento

Caso não exista qualquer música em processamento, a página exibe uma mensagem a informar que a fila se encontra vazia, como ilustrado na Figura 5.7. Esta abordagem evita confusão e cria uma navegação mais intuitiva, sobretudo em cenários de teste ou períodos de baixa atividade.

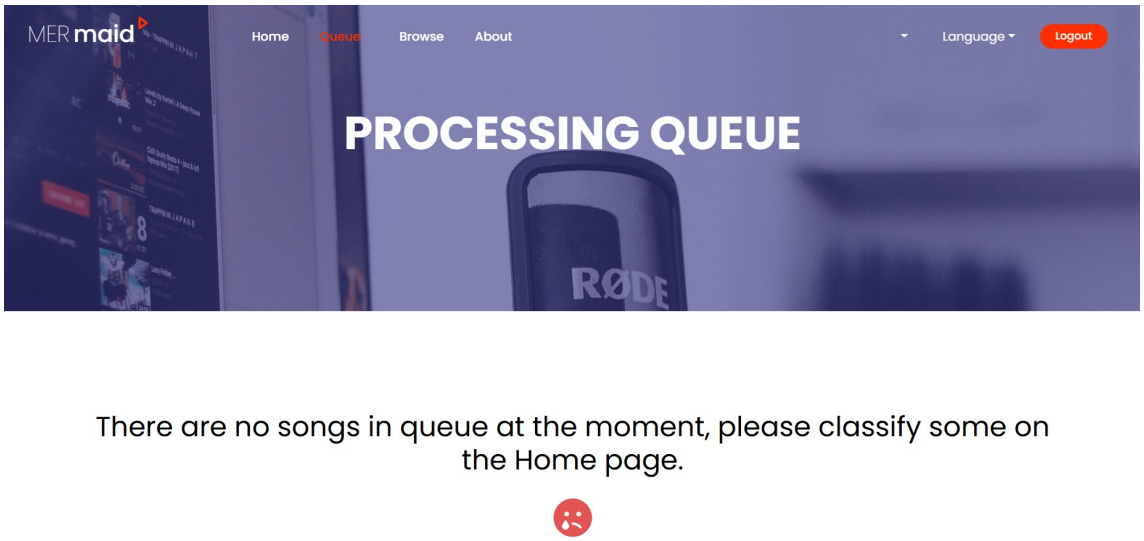

Figura 5.7: Página da fila de processamento vazia

É ainda possível observar a música na página `Browse`, em conjunto com as restantes músicas anteriormente processadas (Figura 5.8), e realizar uma filtragem por nome ou sentimento transmitido, facultando ao utilizador uma camada extra de interação com os resultados.

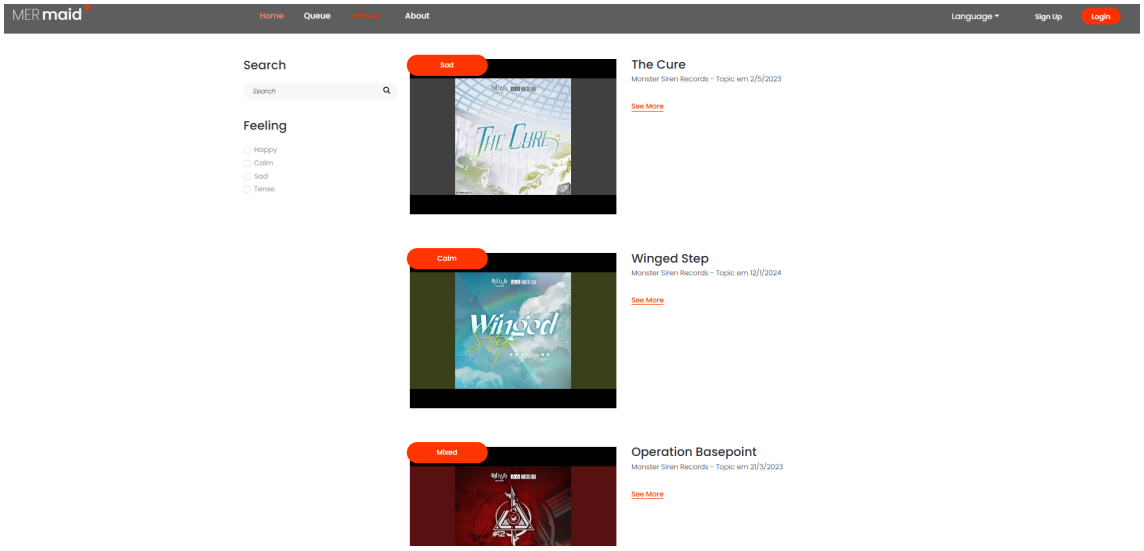

Figura 5.8: Página de navegação da interface Web

Por fim, e com as devidas permissões de administrador, é possível verificar os detalhes de processamento de cada música, como é observável na Figura 5.9, refletindo agora os artefactos reais produzidos, provenientes do *Manager*, em formato acordeão.

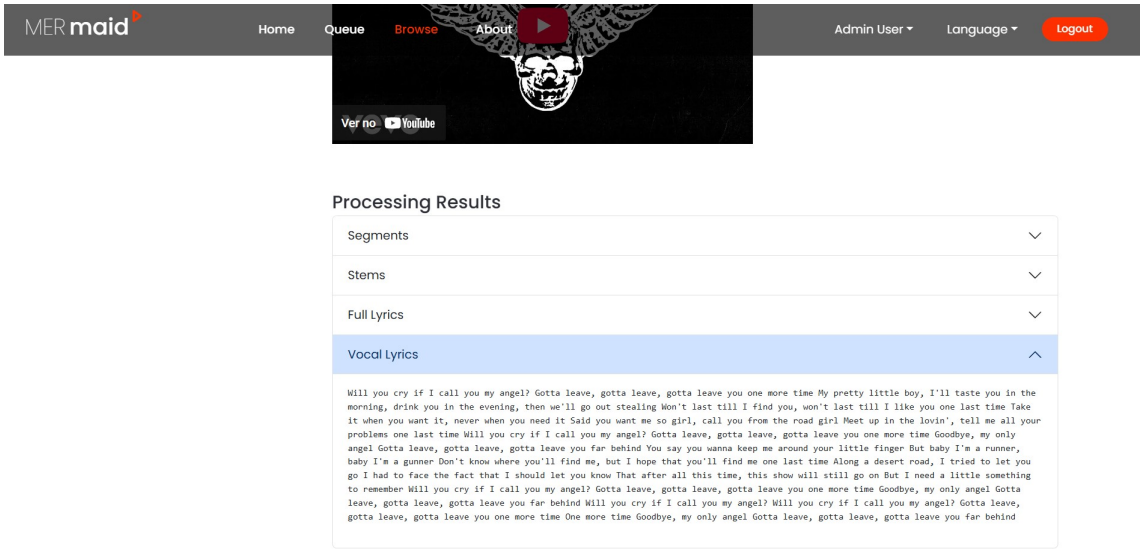

Figura 5.9: Página da música processada e artefactos produzidos

Capítulo 6

Conclusão

Com base no trabalho realizado e em conformidade com os objetivos alcançados, apresenta-se uma reflexão final, bem como propostas para o futuro deste projeto.

6.1 Reflexão

O trabalho desenvolvido permitiu conceber e implementar um sistema modular para processamento musical, baseado numa arquitetura de microsserviços, alinhado com os objetivos propostos na fase inicial do trabalho. A evolução para a versão `MERmaid v0.3` traduziu-se na integração de um conjunto de serviços autónomos de processamento de áudio e no desenvolvimento de uma base sólida e extensível para investigação.

A aposta numa arquitetura distribuída demonstrou ser vantajosa, tanto em termos de escalabilidade como de manutenção, permitindo que cada componente evolua de forma independente. A integração com o RabbitMQ assegurou um fluxo de comunicação assíncrono e resiliente, enquanto que a utilização do MongoDB facilitou uma representação flexível e detalhada de cada etapa de processamento. Embora o foco principal tenha sido criar uma solução funcional e coesa, foi igualmente possível estabelecer uma base sólida para o desenvolvimento futuro. Ao longo do trabalho, mereceram especial atenção os seguintes aspectos: a clareza estrutural; a automatização do ciclo de processamento; a compatibilidade entre módulos escritos em diferentes linguagens; e a correção de diversas inconsistências encontradas.

Naturalmente, ainda existem limitações. A plataforma carece de modelos reais para a classificação de género e emoção, e a API não disponibiliza, no estado atual, um conjunto completo de *endpoints* dedicados à monitorização externa. Ainda assim, estas limitações encontram-se identificadas e enquadradas no plano de evolução previsto, sendo passos lógicos e naturais a implementar numa próxima iteração.

Assim, é possível considerar que o projeto atingiu o objetivo central proposto:

disponibilizar um sistema para processamento musical, baseado em microsserviços, funcional e de fácil manutenção. Ao mesmo tempo, criou a estrutura necessária para que as versões futuras consigam avançar para análises mais complexas, integrando metodologias de investigação musical mais profundas.

6.2 Propostas e Objetivos

A prioridade inicial foi garantir a operacionalidade, estabelecendo uma base estável sobre a qual fosse possível integrar funcionalidades mais avançadas. A estruturação modular adotada facilita a futura substituição de modelos, a expansão funcional e a integração de novas técnicas de análise musical, preservando a organização e o isolamento entre os componentes.

Embora não tenham sido implementados nesta fase todos os serviços e modelos desejados, existe um conjunto claro de extensões planeadas, que podem ser integradas de forma natural, graças à arquitetura desenvolvida. Dentre os serviços e modelos planeados, estão os modelos de classificação emocional, de género, de onda e a possível criação de um modelo de classificação emocional baseado na letra extraída.

A adoção de um *placeholder* no serviço de classificação emocional permitiu validar todo o fluxo de comunicação e garantir que o *Manager*, a API e a aplicação Web já suportam a persistência, o consumo e a representação dos resultados. Quando o modelo definitivo estiver concluído, o serviço passará a produzir resultados fiáveis, sem necessidade de alterar o formato das mensagens, a estrutura de dados ou a interface de visualização.

De forma semelhante, a classificação de género e os módulos de análise visual encontram-se numa fase embrionária, mas com espaço claramente delineado dentro do sistema. A separação de responsabilidades entre microsserviços e a existência de um gestor central com estados bem definidos abrem caminho à introdução de novas etapas, de métricas quantitativas de desempenho e de ferramentas de comparação entre modelos. São exemplos de extensões que podem ser incorporadas sem reestruturar a arquitetura existente: o cálculo automático de medidas, como o WER, a análise temporal do processamento e a inspeção de artefactos intermédios.

O objetivo para a próxima versão, seja ela a `MERmaid v0.4` ou posterior, encontra-se assim claramente traçado: transformar esta base operacional numa ferramenta completa de investigação musical que venha a integrar modelos avançados, métodos rigorosos de avaliação e funcionalidades de exploração e visualização adequadas às necessidades da comunidade científica.

Referências

Aljanaki, A., Wiering, F., & Veltkamp, R. (2015). Emotion based segmentation of musical audio. *International Society for Music Information Retrieval Conference.*

António, T. M. (2021). *MER: Estudo e restruturação de um sistema de reconhecimento emocional em música áudio usando o YouTube.*

Areias, T., & António, T. (2019, 7). *Portal para interação com serviços de reconhecimento emocional em música.*

Cardoso, L., Panda, R., & Paiva, R. P. (2011). *MOODetector: A prototype software tool for mood-based playlist generation.*

Défossez, A. (2021, 11). *Hybrid spectrogram and waveform source separation.* Retrieved from `https://arxiv.org/pdf/2111.03600`

Ekman, P., Friesen, W. V., O'Sullivan, M., Chan, A., Diacoyanni-Tarlatzis, I., Heider, K., … Tzavaras, A. (1987). Universals and cultural differences in the judgments of facial expressions of emotion. *Journal of Personality and Social Psychology*, *53*(4), 712–717. doi: 10.1037/0022-3514.53.4.712

Fernandes, T., & Neves, A. (2024). *Projeto final MERmaid web app v0.2.*

Grilo, J. P. D., & Simões, R. M. (2022, 7). *MOODoke: Emotion-based karaoke.*

Guo, X. (2023). The evolution of the music industry in the digital age: From records to streaming. *Journal of Sociology and Ethnology*, *5*. doi: 10.23977/jsoce.2023.051002

Hennequin, R., Khlif, A., Voituret, F., & Moussallam, M. (2020). Spleeter: a fast and efficient music source separation tool with pre-trained models. *Journal of Open Source Software*, *5*(50), 2154. Retrieved from `https://doi.org/10.21105/joss.02154` (Deezer Research) doi: 10.21105/joss.02154

Louro, P. L., Redinho, H., Malheiro, R., Paiva, R. P., & Panda, R. (2024, 4). A comparison study of deep learning methodologies for music emotion recognition. *Sensors*, *24*. doi: 10.3390/s24072201

Louro, P. M. L. (2023). *Feature engineering and learning for audio-based music emotion recognition.*

Malheiro, R., Panda, R., Gomes, P., & Paiva, R. P. (2018, 4). Emotionally-relevant features for classification and regression of music lyrics. *IEEE Transactions on Affective Computing*, *9*, 240-254. doi: 10.1109/TAFFC.2016.2598569

Malloch, S., & Trevarthen, C. (2018, 10). The human nature of music. *Frontiers in Psychology*, *9*, 364981. Retrieved from `www.frontiersin.org` doi: 10.3389/FPSYG.2018.01680/BIBTEX

McFee, B., Raffel, C., Liang, D., Ellis, D., McVicar, M., Battenberg, E., & Nieto, O. (2015). librosa: Audio and music signal analysis in python. *SciPy*, 18-24. doi: 10.25080/MAJORA-7B98E3ED-003

Panda, R., Malheiro, R., & Paiva, R. P. (2023, 1). Audio features for music emotion recognition: A survey. *IEEE Transactions on Affective Computing*, *14*, 68-88. doi: 10.1109/TAFFC.2020.3032373

Panda, R. E. S. (2019). *Emotion-based analysis and classification of audio music.*

Ribeiro, H. (2023). *Projeto final MERmaid web app.*

Russell, J. A. (1980). A circumplex model of affect. *Journal of Personality and Social Psychology*, *39*(6), 1161–1178. Retrieved from `http://content.apa.org/journals/psp/39/6/1161` doi: 10.1037/h0077714

Virtanen, T. (2007, 3). Monaural sound source separation by nonnegative matrix factorization with temporal continuity and sparseness criteria. *IEEE Transactions on Audio, Speech and Language Processing*, *15*, 1066-1074. doi: 10.1109/TASL.2006.885253

Yang, X., Dong, Y., & Li, J. (2018, 7). Review of data features-based music emotion recognition methods. *Multimedia Systems*, *24*, 365-389. doi: 10.1007/s00530-017-0559-4

Apêndices

Apêndice A

Conjunto de Logs do Serviço Manager

```
Pipeline Manager Service Starting...
Connected to MongoDB at mongodb:27017

HEALTH CHECK STATUS
RabbitMQ:  Connected
MongoDB:   Connected

Service Status:
  download             - 1 consumer(s)
  segmentation         - 1 consumer(s)
  separation           - 1 consumer(s)
  transcription        - 1 consumer(s)

Manager listening on 'manager-requests' queue
Waiting for video URLs...

Received request: https://www.youtube.com/watch?v=nVE1ziLuSNg

STARTING PIPELINE FOR VIDEO
URL: https://www.youtube.com/watch?v=nVE1ziLuSNg

Error creating video entry: duplicate key error (video already exists)
Updated video status to 'processing'

------------------------------
STAGE 1/5: Downloading video
Message sent to queue 'yt-dlp'
```

```
Found file: Sleep Token - Aqua Regia.mp3
Updated download status to 'completed'

------------------------------
STAGE 2/5: Segmenting audio
Message sent to queue 'audio-segmentation'
Created 2 segments

------------------------------
STAGE 3/5: Separating audio stems
Message sent to queue 'source-separation'
Created stems: bass, vocals, other, drums

------------------------------
STAGE 4/5: Transcribing full audio
Message sent to queue 'lyrics-transcription'
Found transcript: Sleep Token - Aqua Regia_lyrics.txt

------------------------------
STAGE 5/5: Transcribing vocals
Message sent to queue 'lyrics-transcription'
Found transcript: Sleep Token - Aqua Regia_vocals_lyrics.txt

Updated video status to 'completed'

Pipeline completed successfully

Processing Summary:
  download                  - completed
  segmentation              - completed
  separation                - completed
  transcription_full        - completed
  transcription_vocals      - completed
```

Excerto A.1: Resumo simplificado da execução do Pipeline Manager

Apêndice B

Elementos da Página Principal da Interface Web

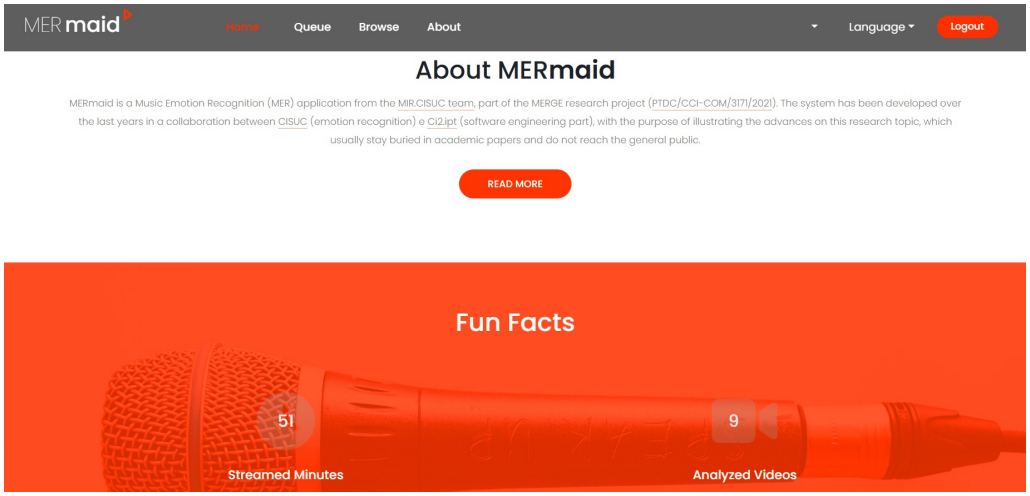

Figura B.1: Secção de factos interessantes da página principal

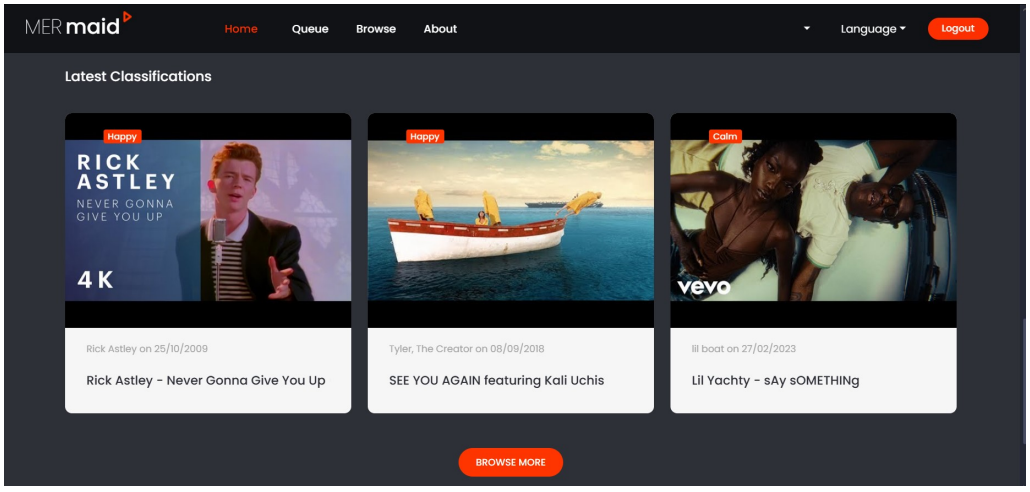

Figura B.2: Secção de classificações mais recentes da página principal

Esta página foi intencionalmente deixada em branco.

ipt Instituto Politécnico de Tomar