Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN ENXEÑARÍA DO SOFTWARE

Aplicación web para a busca de voz en vídeos

Estudante: Roberto Insua Brandariz
Dirección: Paula López Otero
Daniel Valcarce Silva
Javier Parapar López

A Coruña, setembro de 2020.

Á xente que me fai sorrir.

Agradecementos

En primeiro lugar, grazas aos directores pola axuda e paciencia ao longo destes meses. Grazas tamén aos meus amigos, polo seu constante apoio. Grazas aos meus compañeiros, en especial aos *subdesarroladores*, por facer máis amenos estes anos.

Grazas á miña familia, en especial a meus pais, por todo. Todo, durante moitos anos.

Resumo

A aplicación web desenvolvida neste proxecto tratará de proporcionar unha maneira de realizar procuras en coleccións de vídeos pouco explorada até o de agora: buscar no contido das conversas. Para conseguir dita funcionalidade, realizarase a extracción do audio dos vídeos subidos polos usuarios da web. A continuación, o audio será procesado por un recoñecedor de voz para posteriormente gardar a transcrición nun índice invertido. Tras todo este proceso, será posible escribir ou ditar as palabras que se desexen buscar, mostrando os vídeos correspondentes. Ademais, os usuarios disporán da posibilidade de visualizar unha lista de vídeos previamente marcados como favoritos, e outra dos que dito usuario aportou á colección. Por outro lado, cada busca realizada por un usuario rexistrado quedará gardada para a súa posterior consulta. O control do procesado dos vídeos será realizado por un administrador, que deberá decidir o momento no que se debe executar esa pesada operación, para interferir o mínimo posible na dispoñibilidade dos recursos da máquina na que se execute a aplicación.

Abstract

The web application developed on this project will try to provide a, until now, almost unexplored way to search on video collections: search on the content of video conversations. In order to achieve this functionality, the audio will be extracted from the uploaded videos of web users. Then, the audio will be processed by a speech recogniser and the transcription will be saved in an inverted index. After all this process, it will be possible to type or dictate the wanted words to search, showing the matched videos. In addition, users will be able to view a list of videos previously marked as favourites, and another of those uploaded by the user. On the other hand, each search performed by a registered user will be saved for later reference. The control of the processing of the videos will be controlled by an administrator, who will have to decide when that heavy operation should be performed, in order to maximize the availability of resources of the machine.

Palabras chave:

- API
- Eclipse
- FFmpeg
- Git
- GitLab
- Hibernate
- HSQLDB
- HTML
- Java

- JavaScript
- Jenkins
- JPA
- JQuery
- Kaldi
- Lucene
- Materialize
- Maven
- ORM
- PostgreSQL
- Scrum
- Spring
- Spring Boot
- SonarQube
- Taiga
- Thymeleaf
- Web Speech API

Índice Xeral

Índice de Figuras

Índice de Táboas

Capítulo 1

Introdución

Neste capítulo da memoria exponse a motivación, obxectivos e o plan de traballo para o desenvolvemento da aplicación ***VideoBusca***.

1.1 Motivación

Na actualidade, existe unha gran demanda de contidos multimedia por parte dos usuarios. Só en *YouTube*, visualízanse cada mes, máis de 3.000.000.000 horas, e realízanse o mesmo número de buscas. Para satisfacer esta enorme demanda, engádense 500 horas de vídeo por minuto. Dada esta tamén enorme oferta, a busca do vídeo que realmente precisamos pode non ser doada, e é habitual acabar visualizando un vídeo que non trata sobre o tema desexado. Isto ocorre en parte debido a que as procuras realízanse sobre atributos que non describen correctamente o contido do vídeo, como por exemplo, títulos pouco descritivos ou exaxerados expresamente para actuar como *ciberganchos* ou *clickbait*. Porén, nada describe máis o contido dun vídeo que o que se fala nel, polo menos en canto ás conversas que se producen.

1.2 Obxectivos

Tendo en mente os problemas mencionados no apartado anterior, un mercado colosal en pleno crecemento e o descoñecemento de alternativas con propostas similares, nace a idea da creación de ***VideoBusca***. Neste proxecto plantexamos a construción dun buscador de vídeos, o cal permita atopar os resultados a partir das conversas que estes conteñen, coa intención de mellorar a experiencia do usuario.

1.2.1 Obxectivos académicos

Os principais obxectivos académicos a alcanzar no desenvolvemento do proxecto son:

- Poñer en práctica e consolidar os múltiples teóricos adquiridos durante a realización do grado en enxeñería informática.
- Emprego de tecnoloxías e ferramentas non coñecidas até o momento, que requiren a capacidade de *aprender a aprender*, que é unha parte implícita neste grado.
- Utilización dunha metodoloxía clave actualmente no sector do desenvolvemento *software*.

1.2.2 Obxectivos da aplicación

Os principais obxectivos a alcanzar no desenvolvemento da aplicación son:

- Transcrición do audio dunha colección de vídeos a texto mediante un recoñecedor de voz.
- Indexación de dita transcrición para posibilitar unha procura eficiente.
- Implementación dun buscador para a procura de vídeos en base a esa transcrición.
- Emprego dun recoñecedor de voz para ditar as palabras a utilizar no buscador.
- Implementación do rexistro e autenticación de usuarios para a aplicación.
- Implementación dun sistema que permita aos usuarios subir os seus vídeos, que se engadirán á colección dispoñible para a súa busca.
- Permitir aos usuarios marcar como favoritos os vídeos desexados, e a súa posterior visualización.
- Visualización por parte do usuario do seu historial de procuras.
- Implementación dun panel de administración para poder controlar o momento no que realiza o procesado dos vídeos.

1.3 Estrutura da memoria

A memoria do presente proxecto está estruturada da seguinte maneira:

- **Introdución**: describe a motivación e o contexto no que se realiza o proxecto. Tamén explica os obxectivos a conseguir e a estrutura da memoria.
- **Estado do arte**: realiza un estudo de alternativas á aplicación desenvolvida neste proxecto.

- **Conceptos**: explica os conceptos tecnolóxicos sobre os que se basea a aplicación.
- **Tecnoloxías e ferramentas**: describe as tecnoloxías e ferramentas empregadas para o desenvolvemento deste proxecto, xunto coa xustificación da súa elección
- **Proceso de enxeñería**: detalla a metodoloxía elixida e a xestión do proxecto realizada.
- **Desenvolvemento**: describe a análise de requisitos e a arquitectura proposta para o desenvolvemento, xunto co camiño de datos utilizado no procesado e busca dos vídeos.
- **Implementación**: explica como se desenvolveron as partes claves da aplicación *Java*.
- **Conclusións**: presentase a situación final do proxecto, as liñas aprendidas e as posibles liñas futuras de traballo.

Capítulo 2

Estado del arte

Neste capítulo veremos alternativas similares ao que se propón no proxecto: buscadores de vídeos, unha ferramenta de recoñecemento de voz e os asistentes virtuais, que permiten realizar buscas coa voz.

2.1 Buscadores de vídeos

Para realizar buscas en coleccións de vídeos, existen moitas alternativas na actualidade, como *YouTube*[1] ou *Vimeo*[2]. Estas páxinas xeran cantidades enormes de tráfico na rede. Aínda que non se poden comparar en complexidade ou tamaño coa aplicación realizada neste proxecto, si que podemos observar como teñen solucións distintas para problemas compartidos.

2.1.1 Buscador

Os buscadores de estas dúas páxinas posúen gran cantidade de opcións para filtrar os resultados. En definitiva, buscan polos atributos do vídeo: título, descrición, data de subida, duración, etc. Pola contra, o buscador deste proxecto utiliza unicamente os datos recompilados polo recoñecedor de voz.

Porén, *YouTube* xa ten un recoñecedor de voz integrado, como veremos a continuación. É posible que nun futuro combine ámbalas fontes de información para mellorar os resultados.

2.1.2 Recoñecemento de voz

Youtube utiliza o recoñecemento de voz, aínda que de maneira distinta á de *Videosearch*, xa que o utiliza para a navegación e a creación de subtítulos.

[1] https://youtube.com
[2] https://vimeo.com

Navegación

A aplicación móbil de *YouTube* dispón dun recoñecedor de voz, o cal permite a navegación mediante comandos ou a busca ditándolle as palabras desexadas.

Subtítulos

Unha das opcións dos subtítulos de *YouTube* é a xeración automática destes, o que se produce cun recoñecedor de voz que utiliza redes neurais profundas, ao igual que os utilizados neste proxecto. De feito, o recoñecedor de voz que utilizamos para buscar dende a web, o *Web Speech Api*, tamén utiliza o recoñecemento de voz de *Google* (propietaria de *YouTube*) cando é usado desde *Google Chrome*.

Por outro lado, ditos subtítulos poden ser traducidos automaticamente ou ser aproveitados por ferramentas de terceiros, como se comenta no seguinte apartado.

2.1.3 Busca nas conversas dos vídeos

Aínda que non estea soportado directamente, si que é posible realizar este tipo de buscas en *YouTube*. A inclusión dos subtítulos, tanto de maneira manual como xerados automticamente, permitiu a creación de ferramentas que permiten buscar o momento exacto dun vídeo a partir dunha frase ou palabra. Dita funcionalidade sería unha magnífica inclusión de cara a futuro neste proxecto.

Exemplos destas ferramentas son as extensións de *Google Chrome*: *Keyword Search in YouTube Video* [3] e *Invideo* [4].

2.2 Recoñecedores de voz ASR

Durante moito tempo, o recoñecedor de voz considerado *estado do arte* era *HTK*[5]. Trátase dun kit de ferramentas para manipular modelos ocultos de Markov (*HMM*), que está pensado principalmente para recoñecemento da fala. Está implementado en *C* e é gratuíto, pero conta cunha licenza propietaria que se debe aceptar.

HTK e *Kaldi* son algúns dos kits de ferramentas para recoñecemento de voz de máis sona na actualidade. Comparativamente, contan con bastantes diferenzas [14]:

- *Kaldi* inclúe funcionalidades e *scripts* que serían complicados de implementar en *HTK*, mentres que facelo ao contrario, podería ser máis sinxelo.

[3] https://chrome.google.com/webstore/detail/keyword-search-in-youtube/dpadfnjbfalopginhcmmmccpfdaofjco/related

[4] https://chrome.google.com/webstore/detail/invideo-for-youtube/iacbjlffnpbhgkgknabhkfmlcpdcigab

[5] http://htk.eng.cam.ac.uk/

- *Kaldi* está por defecto preparado para realizar certas tarefas, as cales se deberían desenvolver por completo en *HTK*, como por exemplo a preparación dos datos, executar o adestramento, obter e avaliar os resultados...
- *HTK* conta con decodificadores (*HDecode* [6]) que realizan a composición do dicionario e dos modelos acústico e lingüístico, mentres que Kaldi non.
- *HTK* conta con *HVite* [7], o que permite procesar gramáticas complexas.
- *Kaldi* utiliza un repositorio git [8] mentres que *HTK* non conta cun repositorio público.
- Actualmente *Kaldi* posúe unha comunidade máis activa.

En algunhas comparacións, poñen en gran consideración a *Kaldi* respecto a *HTK*, con frases como: *As receitas completas que prové Kaldi producen grandes resultados por defecto*, ou tamén: *HTK proporciona menos soporte e precisa de moito máis tempo, coñecemento e esforzo para obter resultados similares a outros kits de ferramentas*[1].

recognizer	VM1	WSJ1
HDecode v3.4.1	22.9	19.8
Julius v4.3	27.2	23.1
pocketsphinx v0.8	23.9	21.4
Sphinx-4	26.9	22.7
Kaldi	12.7	6.5

Figura 2.1: Taxa de erro de diferentes recoñecedores de voz con datos de proba (VM1, WSJ1). *HTK* é usado xunto cos descodificadores *HDecode* e *Julius* [1]

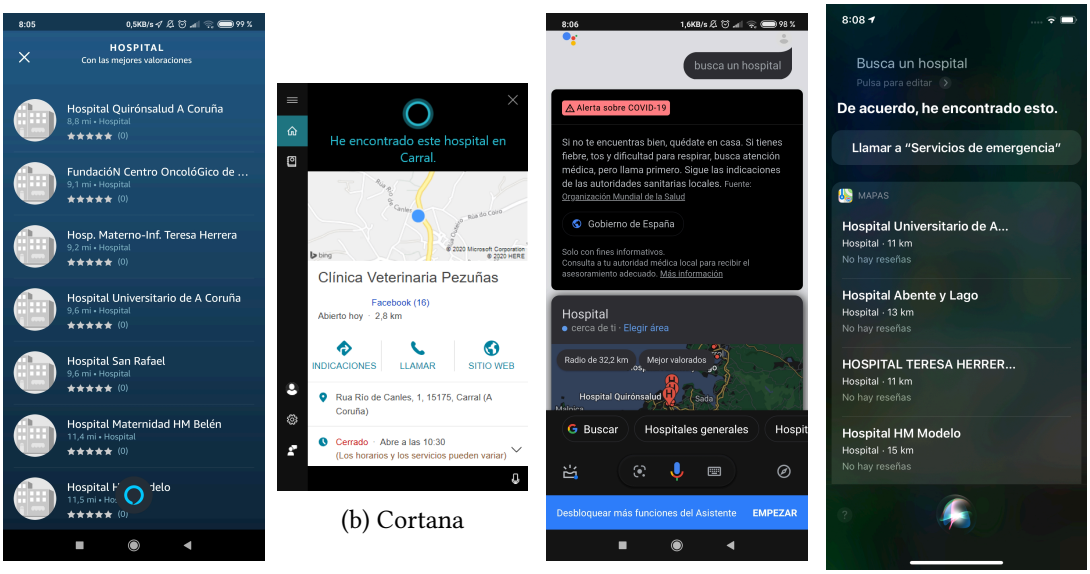
.

2.3 Recoñecemento de voz nos asistentes virtuais

As ferramentas de recoñecemento de voz que os usuarios teñen máis a man son os asistentes virtuais presentes nos ordenadores ou dispositivos móbiles. Ditas ferramentas non só transforman a voz en frases ou palabras, senón que son capaces de interpretar eses datos dunha maneira similar á humana, permitindo unha comunicación máis natural. Dito doutra maneira, estes asistentes virtuais van máis aló do recoñecemento das frases, xa que son capaces de comprender o que busca ou precisa o usuario en cada instante, mediante unha pregunta ou orde.

Entre os asistentes máis coñecidos encóntranse os seguintes:

[6] http://htk.eng.cam.ac.uk/extensions/index.shtml
[7] http://www1.icsi.berkeley.edu/Speech/docs/HTKBook/node278.html
[8] https://github.com/kaldi-asr/kaldi



(a) Alexa (b) Cortana (c) Google Assistant (d) Siri

Figura 2.2: Exemplos de resultados nos principais asistentes virtuais coa petición "Busca un hospital"

- *Siri*[9] é o asistente virtual de *Apple*, presente unicamente nos seus dispositivos. Utiliza *Google* como buscador e un recoñecedor de voz baseado nunha *Deep Neural Newtork* (DNN)[2].

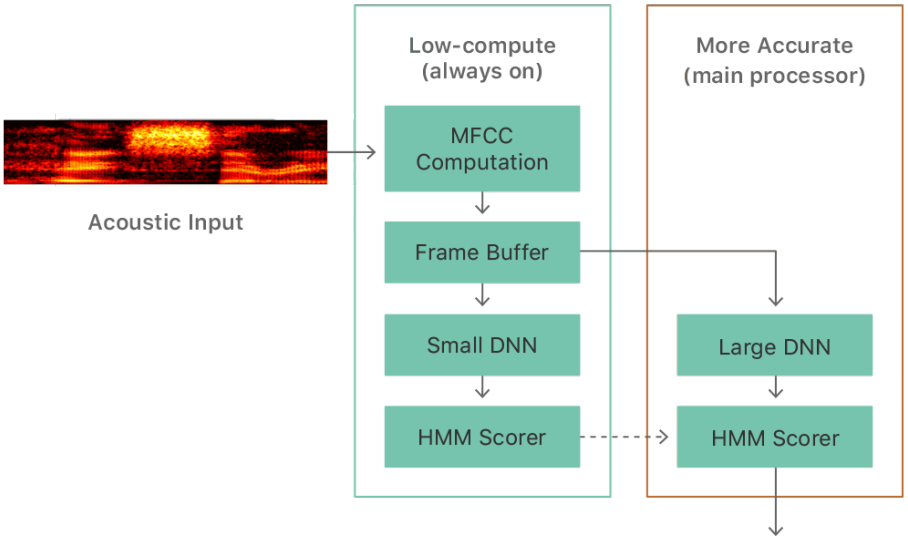


Figura 2.3: Detección en dous pasos de Siri[2]

[9] https://www.apple.com/es/siri/

O seu funcionamento consta de varias partes. Inicialmente executase unha pequena versión da *Deep Neural Newtork*, en busca dunha máxima eficiencia. Cando se detecta bo resultado potencial, volve procesarse coa versión completa da *Deep Neural Newtork*. Tras o visto e prace dado polo dispositivo, a información é enviada á nube, onde se volve a comprobar. Dependendo do resultado, cancelase a acción ou comezan os seguintes pasos do procesamento: comprender que dixo o usuario e proporcionarlle o que necesita.

- *Google Assistant*[10] é, como o seu nome indica, o asistente virtual de Google. Está presente en dispositivos móbiles tanto *iOS* como *Android*, ademais dos altofalantes intelixentes *Google Home*.

  Neste caso, o recoñecemento de voz consta de tres fases: na primeira, o dispositivo detecta a frase iniciadora mediante un sistema de *keyword spotting* (KWS) baseado nunha *Deep Neural Newtork* [15]. Posteriormente, envíanse os datos ao servidor, onde se volve comprobar a frase iniciadora. Se todo é correcto, realizase no servidor o procesado da petición mediante o sistema de recoñecedor de voz, para posteriormente devolver ao dispositivo a resposta adecuada. [3]

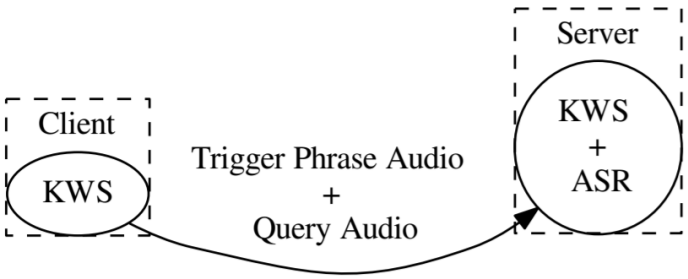


Figura 2.4: Diagrama do sistema de recoñecemento de voz utilizado en Google Asisstant [3]

- *Cortana*[11] pola súa banda, é a aposta de *Microsoft* neste campo. Pode utilizarse en dispositivos con *Windows 10* e na súas consolas. Utiliza *Bing* como buscador.

  En canto ao recoñecemento de voz, este iniciase no dispositivo con un minimizado sistema de *Keyword Spotting* (KWS)[16], que se utiliza para recoñecer a frase iniciadora. En caso de éxito, a petición é examinada na nube por un recoñecedor de voz e por un analizador semántico[17] para comprender que necesita o usuario e enviarlle unha resposta adecuada.

[10] https://assistant.google.com/
[11] https://www.cortana.es/

- *Alexa*[12] é o asistente virtual de *Amazon*. Está dispoñible en *iOS* e *Android*, pero tamén nos altofalantes intelixentes *Amazon Echo*.

  Igual que os demais asistentes virtuais, realiza unha primeira transcrición de voz no propio dispositivo cun sistema *Keyword Spotting* (KWS) [18] [19], principalmente para detectar a frase iniciadora e posteriormente procesa dito audio na nube.

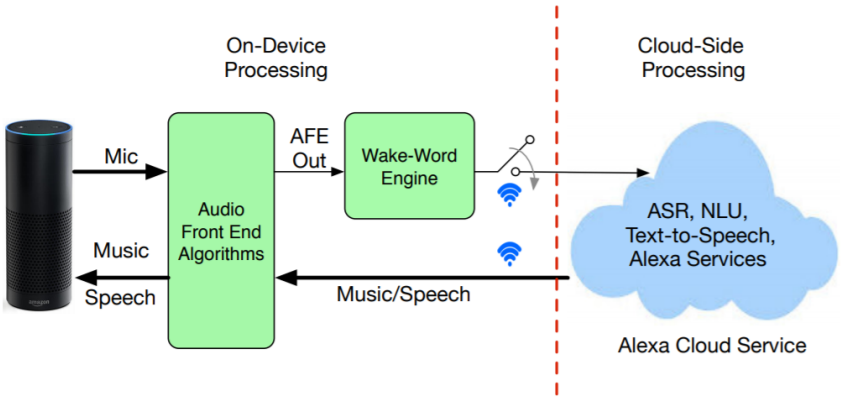


Figura 2.5: Diagrama do sistema de recoñecemento de voz utilizado por Alexa [4]

[12] https://developer.amazon.com/es-ES/alexa

Capítulo 3

Conceptos

Neste capítulo, veremos os conceptos tecnolóxicos sobre os que se sustenta a aplicación.

3.1 Recoñecemento de voz

A base deste proxecto é claramente o recoñecemento de voz. Aínda que non se realiza unha implementación, si que que se utiliza tanto no procesamento dos vídeos como na busca dende a web, polo que cómpre entender o proceso realizado no recoñecemento e adestramento previo do sistema.

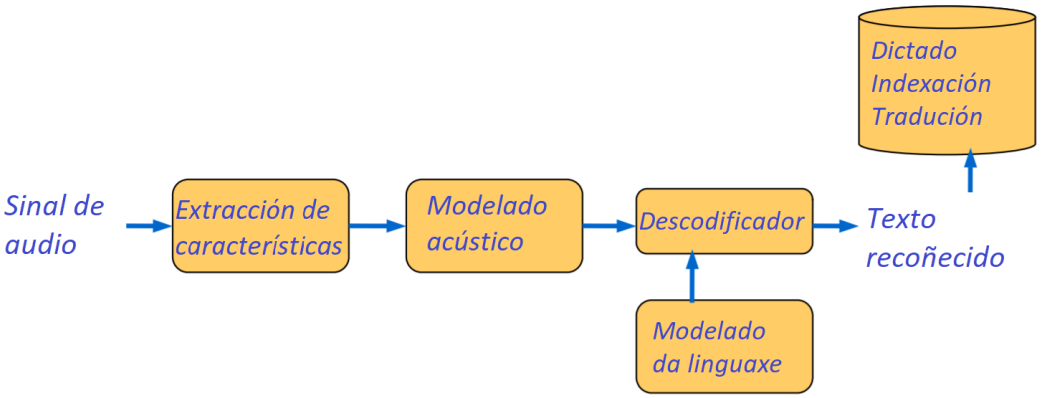
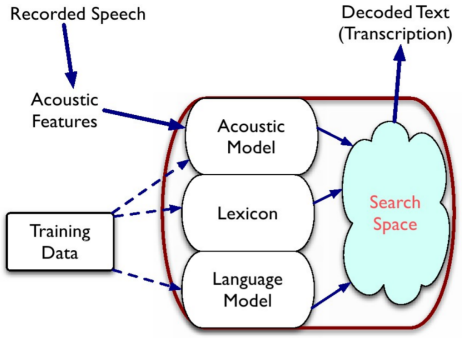


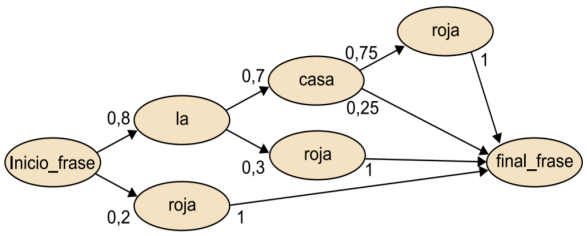
Figura 3.1: Proceso do recoñecemento de voz [5]

3.1.1 Descodificador

Para conseguir a transcrición máis probable para unha sinal acústica, necesitamos combinar o modelo acústico, o modelo lingüístico e o dicionario, creando un espazo de busca para o posterior análise do audio (figuras 3.2a e 3.2b).



(a) Partes necesarias para a descodificación. [5]



(b) Grafo do espazo de busca para o dicionario coas palabras (la, casa, roja) cun modelo de linguaxe 2-grama. En cada transición de estados vemos as probabilidades de que suceda. [20]

Figura 3.2: Esquema xeral do sistema e exemplo dun espazo de busca

En base ás probabilidades do modelo de linguaxe e ás producidas pola análise da sinal de audio, calcúlanse "todas" as probabilidades das ramas do grafo do espazo de busca, comunmente mediante o algoritmo de *Viterbi*. Ditos cálculos supoñen unha gran carga de traballo, polo que en sistemas moi complexos limítanse os camiños explorados do grafo con técnicas de *pruning* ou poda.

3.1.2 Extracción de características

O primeiro paso a realizar nun sistema de recoñecemento de voz é a extracción de características da voz, que consiste en obter unha representación do sinal de voz empregando un conxunto de parámetros extraídos de dito sinal.

Para a extracción das características, o método máis común é a obtención dos *coeficientes cepstrais das frecuencias de mel (MFCC)*. Para iso, calcúlase o espectro da frecuencia do sinal, dividíndoo en pequenos intervalos, aplicándolle unha transformada de Fourier e posteriormente calculando as enerxías espectrais. Neste momento, a sinal contén moita información, parte de ela redundante e mesturada con ruído. Se aplicamos a escala *mel*, acentúamos certas bandas de frecuencias nas que se encontra a voz humana, á vez que diminuímos a importancia doutras que non nos interesan. Con este paso reducimos enormemente a información redundante e o ruído. Finalmente calcúlase o logaritmo de todas as enerxías de cada frecuencia *mel* e comprímese, polo que xa teríamos os *MFCCs*. [5] [21]

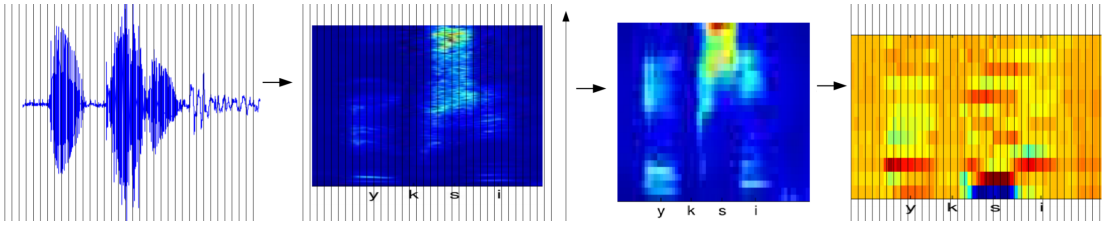


Figura 3.3: Frecuencia de voz ⇒ Espectrograma de enerxía ⇒ Espectrograma *mel* ⇒ Espectrograma *MFCC* (composición de imaxes de [5])

3.1.3 Modelo acústico

O modelo acústico define os sons da linguaxe, pero tamén un modelo estatístico para determinar a probabilidade de que estes ocorran.

Unidades acústicas

Para a posterior formación de palabras, necesitamos obter as unidades acústicas coñecidas como fonemas, tratando de ignorar outros sons que poidan existir no modelo acústico.

Os fonemas son as partes que compoñen as palabras e, á súa vez, divídense principalmente en vocais (o fluxo de aire non é obstruído) e consoantes (o fluxo de aire está obstruído parcial ou totalmente) [5] . Ao tratalos no recoñecemento de voz hai que ter en conta diversos puntos:

- Cada linguaxe utiliza un conxunto distinto de fonemas.
- Os fonemas anterior e posterior poden introducir variacións na pronunciación (coarticulación).
- A maneira de pronunciar: velocidade de pronunciación, dialectos, ton da voz...

Porén, o máis común nos modelos acústicos non é utilizar fonemas *simples*, senón *trifonemas*. Combinando o fonema actual co anterior e posterior, obtemos un trifonema, polo que pasamos de utilizar fonemas independentes a outros dependentes do contexto no que se encontran, axudando a controlar a coarticulación [20].

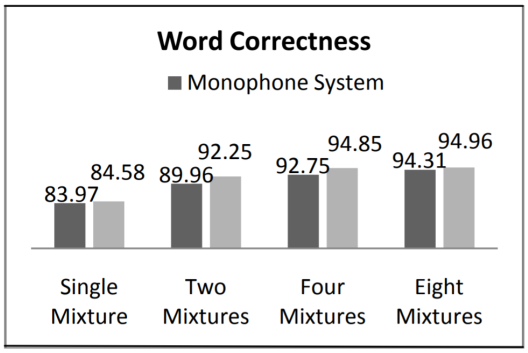


Figura 3.4: Comparativa das porcentaxes de acerto entre sistemas con fonemas simples e con trifonemas [6]

Estimación e modelo estatístico

Para o modelo estatístico o máis común é utilizar un *modelo oculto de Markov* ou *HMM*, o cal proporciona unha forma robusta de cuantificar os patróns da fala. A fala é descrita como modelos estatísticos parametrizados e o recoñecemento realízase considerando cal é o modelo con máis probabilidades de que se produza dada a secuencia de observación actual [6].

Un *HMM* está formado por tres elementos: estados, transicións e símbolos emitidos. Nel, o estado actual permanece oculto e só é posible acceder á información emitida, a cal lle foi proporcionada no paso anterior. Cada estado representa un son característico e corresponde a un subfonema ou fonema simple. Ademais, débense ter en conta tanto a probabilidade de transición entre estados, como as probabilidades de emisión dos símbolos [20], o que se calcula mediante un *modelo de mesturas gaussianas* ou *GMM*.

Con todo isto, podemos realizar diversas operacións:

- Recoñecemento: dado un modelo e un sinal observado, encontrar a secuencia de estados máis probable. Utilízase o algoritmo de Viterbi, que é un método de busca en grafos con pesos.
- Adestramento: dadas varias sinais observadas, modificar os parámetros do modelo. Pódese utilizar o algoritmo de *Baum-Welch* (tamén coñecido como *forward-backward*), o cal produce un modelo de recoñecemento flexible ante sinais de son pouco óptimas [5].
  Para unha correcta estimación do modelo acústico, o ideal sería obter gravacións de distintas persoas e con diversas variacións: distinto ruído ambiente, cambios voluntarios no ton da voz, etc. Unha vez procesadas, o modelo poderá contar con varias variacións de cada trifonema o que lle permitirá gardar uns datos fiables.

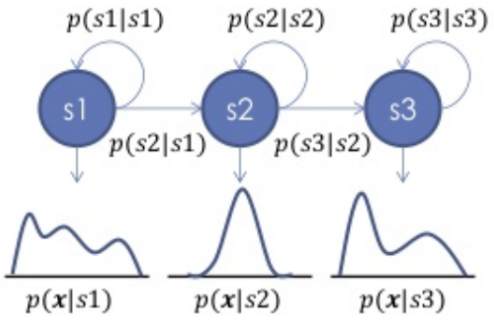


Figura 3.5: Representación dun *HMM* dun trifonema (arriba) e os seus correspondentes sons respecto a o modelo *GMM* (abaixo) [7]

3.1.4 Modelo da linguaxe

O modelo da linguaxe define as palabras e a probabilidade de que aparezan xuntas unhas coas outras, tendo en conta as regras propias do idioma. O máis común para estes cálculos é utilizar *N-gramas*, polo que só se utilizan as *N* palabras anteriores para estudar a aparición da seguinte palabra da frase.

Canto maior sexa o valor de N, maior deberá ser o número de mostras para adestrar o sistema, escalando de maneira exponencial. Pese a contar cunha gran base de datos, non existirán todos os N-gramas, imposibilitando que sexan recoñecidos cando dita secuencia teña probabilidade 0.

Para evitar que eses teñan unha probabilidade de 0, existen solucións ao problema, entre as cales destaca o *suavizado de Kneser-Ney*. Como podemos observar na figura (ref:Kneser-Ney), esta técnica busca coincidencias para un N-grama e aplícalles un "suavizado" na probabilidade para evitar que esta sexa demasiado alta. No caso de que dito N-grama non se atope, buscaría un (N-1)-grama, (N-2)-grama... ata que atope unha coincidencia ou chegue a un 1-grama. En ambos casos daríalle ao N-grama certa probabilidade para que poida participar no proceso de recoñecemento, pero esta será menor canto máis pasos retroceda.

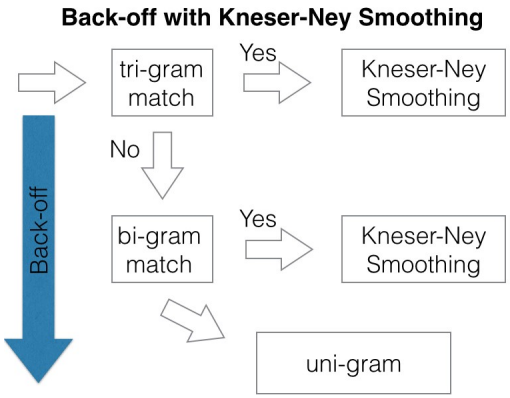


Figura 3.6: Esquema do suavizado *Kneser-Ney* [8]

Dicionario

O dicionario é o conxunto de vocabulario utilizado polo sistema, xunto coa súa pronunciación. Cada palabra pode ter varias secuencias de fonemas dependendo das diversas maneiras de pronuncialas. Ao crear un dicionario débese ter en conta o que se busca co sistema de recoñecemento de voz: se é pequeno faltaranlle moitas palabras, mentres que un dicionario grande pode inducir a erro pola gran cantidade de posibilidades.

```
rompeu rompeu R_B o_I m_I p_I e_I u_E
rompeu rompeu R_B o_I m_I p_I e_I w_E
rompeuna rompeuna R_B o_I m_I p_I e_I w_I n_I a_E
rompeuse rompeuse R_B o_I m_I p_I e_I u_I s_I e_E
rompeuse rompeuse R_B o_I m_I p_I e_I w_I s_I e_E
```

Figura 3.7: Exemplo do dicionario de *Kaldi* para galego utilizado na aplicación. Aparecen destacadas as variacións de pronunciación para dúas palabras.

3.2 Recuperación da información

A indexación é un proceso para a ordenación de datos de maneira que se facilite a súa consulta e análise. Dada a necesidade dun método eficiente para buscar e acceder a grandes cantidades de datos, comezaron a aparecer os *índices invertidos*. O compromiso a cambio de buscas rápidas, é o procesado que se debe facer dos arquivos antes de engadilos á colección de datos.

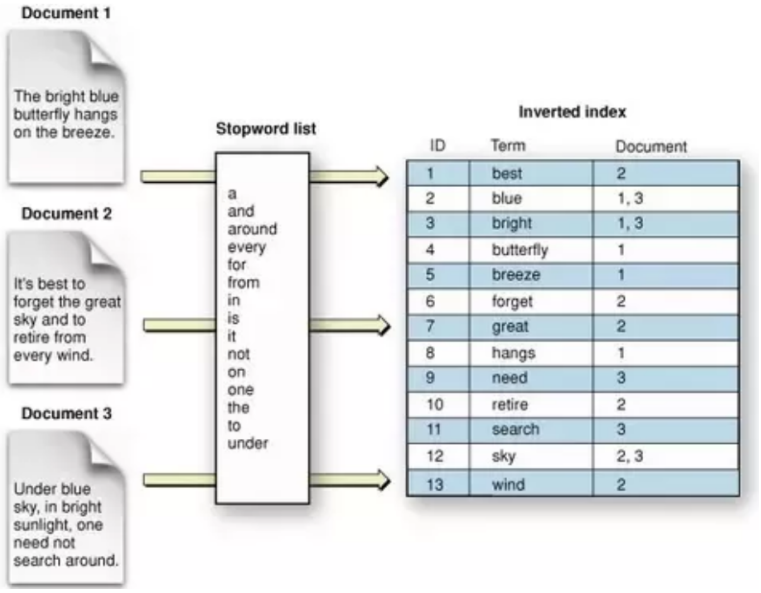


Figura 3.8: Simplificación da creación dun índice invertido. [9]

Para construír un índice invertido relativo a buscas de texto son necesarios os seguintes pasos:

- Recolección dos documentos a indexar.
- Dividir a información por palabras e eliminar caracteres innecesarios como os puntos ou as comas. É unha práctica bastante estendida a eliminación das palabras baleiras ou *stopwords* que aportan pouco contido á busca, aínda que en algúns casos isto non se realiza, co obxectivo de poder recoñecer unha linguaxe máis natural nas buscas.
- Procesado das palabras para normalizalas, o que consiste en pasar todo a minúsculas e aplicar *stemming* ou redución á raíz. Desta maneira obtense o termo base da palabra, polo que as futuras buscas devolverán máis resultados e evitamos por exemplo perder un resultado por culpa de diferentes conxugacións dun verbo.
- Indexación de cada termo nun índice cunha referencia aos documentos nos que se atopou.

Posteriormente, para a busca, simplemente deberíamos buscar a palabra desexada e obteríamos os resultados dos documentos nos que se encontra para así poder procesalos ao noso gusto. [22]

3.2.1 Recuperación de documentos da fala

O coñecido como *Spoken document retrieval (SDR)* pretende mellorar a recuperación da información contida en documentos que son resultado dunha transcrición cun recoñecedor da fala. En dita transcrición é posible atopar un alto ratio de erros, polo que non se debe tratar como unha tarefa típica de recuperación da información, xa que un nivel tan alto de ruído resultará nunha mala efectividade [23]. Se implementamos un sistema *Query-by-Example Spoken Document Retrieval (QbESDR)*, este ruído existirá tamén na entrada de datos relativos á consulta, xa que será necesario transcribir a frase pronunciada polo usuario [24].

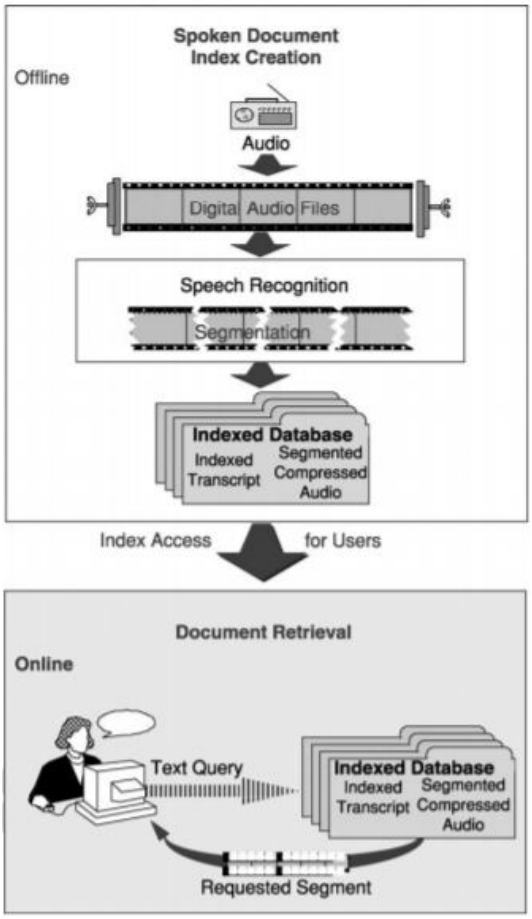


Figura 3.9: Arquitectura básica dun sistema *Spoken document retrieval (SDR)* [10].

Sendo o obxectivo a recuperación da información dentro do documento, e non o documento en si mesmo, é necesario gardar no índice invertido o instante no que se pronuncia cada termo. Desta maneira, o usuario poderá consultar directamente o resultado da súa busca, sen ter que reproducir todo o documento.

Por outro lado, a integración de solucións *QbESDR* é inviable en moitos casos, xa que tenden a ser eficaces, pero pouco eficientes, impedindo unha experiencia de usuario rápida e sen esperas [25]. Ademais, é desexable que as buscas permitan certa variación, xa que no caso de buscar unha frase longa, é bastante improbable que exista un resultado exacto. En cambio, é máis probable que existan resultados semellantes, ou cando menos, con certa relación co tema da consulta e que poden ser interesantes para o usuario.

Capítulo 4

Tecnoloxías e ferramentas

Neste capítulo, veremos as tecnoloxías e ferramentas coas que se construíu a aplicación, xunto con algúns detalles sobre o seu uso.

4.1 Recoñecemento de voz

Nesta sección descríbense as tecnoloxías de recoñecemento de voz utilizadas na aplicación e explícanse as decisións tomadas sobre o seu uso.

4.1.1 Kaldi

Kaldi [1] é un kit de ferramentas gratuíto para recoñecemento de voz distribuído baixo a *Apache License*. Está escrito en *C++*. Xa falamos del no estado do arte 2.2, pero dado que o recoñecedor de voz sobre o que se basea o proxecto está construído con *Kaldi*, completamos a información coas súas características máis destacadas:

- Deseño extensible grazas á implementación de algoritmos xenéricos.
- Soporte para álxebra lineal.
- Optimizado para computación paralela.
- Modelos de adestramento na *GPU*.
- Compatible coa identificación do falante, polo que pode ser utilizado para autenticar ou verificar a súa identidade.

[1] https://kaldi-asr.org/

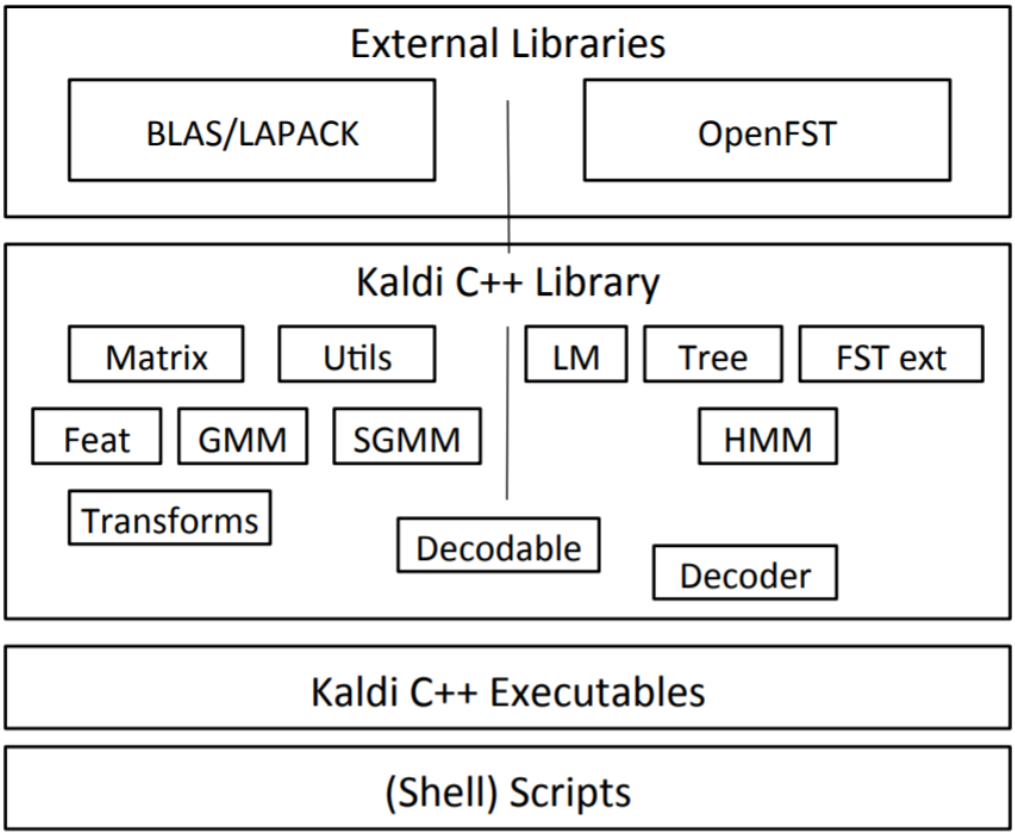


Figura 4.1: Arquitectura de Kaldi [11]

Arquitectura dos modelos de recoñecemento de voz empregados

Neste proxecto, utilizouse un recoñecedor da fala continua con vocabulario extenso ou *LVCSR*, construído empregando *Kaldi* nunha colaboración entre o Grupo de tecnoloxías multimedia (GTM) [2] da Universidade de Vigo e o Information Retrieval Lab (IRLab) [3] da Universidade da Coruña [26].

En resumo, para o seu desenvolvemento foron utilizados modelos acústicos baseados nunha *Deep neural Network (DNN)* dependentes do contexto, adestradas mediante o enfoque de Karel Vesely's, tal e como se explica en [27].

As características das sinais de entrada para os modelos acústicos son uns *MFCCs* de 40 dimensións. Cada fragmento de voz é concatenado cos fragmentos que o rodean e posteriormente pasan por cada unha das 6 capas coas que conta a *DNN*, sendo o resultado de saída unha representación dos estados HMM.

O descodificador *LVCSR* xera un grafo de palabras, o cal é procesado e indexado [28] de

[2] http://gtm.uvigo.es/
[3] https://www.dc.fi.udc.es/ir/index.html

tal maneira que se crea un índice invertido de todas as frases do grafo.

No caso das palabras que non estean no vocabulario (*OOV*), búscase a palabra máis similar foneticamente mediante *proxies* [29] e realízase a busca con esa palabra.

O adestramento dos modelos acústicos do *LVCSR* foi realizado, no caso do idioma galego, mediante a base de datos de noticias en galego Transcrigal [4], eliminando previamente todas as partes sen voz ou con erros de pronunciación, frases incompletas ou cortas. Por outro lado, para a creación do modelo lingüístico foron adestrados dous modelos baseados en 4-gramas, cada un con distintas fontes de datos, aos que se lles aplicou o suavizado de Kneser-Ney. A partir da mestura destes dous modelos lingüísticos, obtívose o modelo final, ao que se lle limitou o tamaño ás 300.000 palabras máis frecuentes.

Emprego no proxecto

Inicialmente considerouse a utilización dunha interface xa existente para *Java*[5] que puidese chamar ao recoñecedor de voz. Esta opción foi descartada debido á falta de documentación, a que o proxecto non contaba con actividade recente e aos múltiples fallos de incompatibilidade que se producían ao intentar integralo. A cambio, decidiuse executar os arquivos *.sh* de *Kaldi* dende o propia aplicación *Java*.

As consideracións a ter en conta son as seguintes:

- Para o emprego do sistema de recoñecemento da fala, necesitamos arquivos de audio con formato *.wav*, unha soa canle de son e 16kHz.
- Débese ter en conta que os arquivos de resultados Kaldi están codificados en *ISO-8859-1*, mentres que o entorno está configurado para utilizar *UTF-8*.
- O procesado do recoñecedor de voz consume unha gran cantidade de recursos, polo que se necesita un ordenador cunha boa *CPU* (a cantidade de núcleos utilizados é configurable) e, sobre todo, gran cantidade de memoria *RAM* libre. Un *SSD* tamén supón unha enorme mellora.

4.1.2 Web Speech API

Para a busca de vídeos, podemos utilizar texto ou voz. No caso do recoñecemento de voz, utilizamos a *Web Speech Api*. Como o seu nome indica, é unha *API* para o recoñecemento da fala, á cal debe accederse mediante *JavaScript*. En 2012, a *W3C Community* [6] presentou a

[4] https://pdfs.semanticscholar.org/acbf/2b74dcc2fc49ab8a2748264a6a8f6b073410.pdf

[5] https://github.com/danijel3/KaldiJava

[6] https://www.w3.org/community/

especificación que debería cumprir dita *API*. Anos despois, practicamente só se pode usar con *Google Chrome*[7], dado que é o único navegador maioritario que a implementou, utilizando o recoñecemento de voz propio de Google [30]. Pese a isto decidiuse a súa utilización debida á abstracción que proporciona, permitindo obviar o recoñecedor que traballa por detrás e a tecnoloxía utilizada.

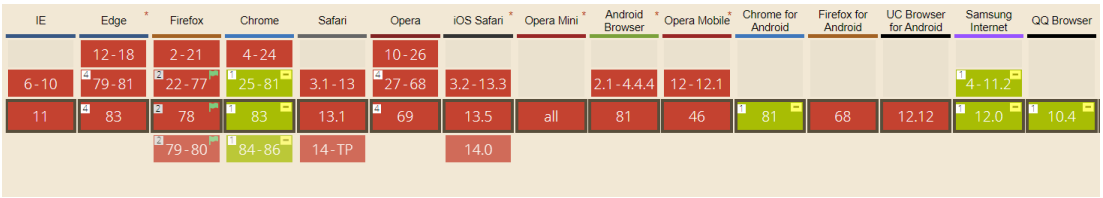


Figura 4.2: Navegadores que soportan a *Web Speech API*. Xullo de 2020 (lista incompleta) [12]

No noso caso, está configurada para que recoñeza o galego, ao igual que Kaldi. Nos test realizados comprobamos que funcionaba en *Google Chrome*, pero fallaba por diversos motivos en *Mozilla Firefox* [8], *Microsoft Edge* [9], *Opera* [10] e mesmo en *Chromium* [11], o cal sorprende ao estar tan relacionado co navegador de *Google*.

Por outro lado, para que o navegador *Google Chrome* permitise o seu funcionamento, era necesario permitir a conexión por *HTTPS*, xa que sen esa capa extra de seguridade, non nos permitía acceder ao micrófono.

4.2 Bases de datos e persistencia

Nesta sección descríbense as tecnoloxías relativas á persistencia de datos utilizadas na aplicación e explícanse as decisións tomadas sobre o seu uso.

4.2.1 PostgreSQL

PostgreSQL[12] é unha base de datos relacional orientada a obxectos que utiliza o estándar *SQL*. Distribúese baixo licenza de código aberto.

As características máis importantes de *PostgreSQL* son [31] [32]:

- Multiplataforma: dispoñible para *Windows* e nun gran número de versións de *Unix*.

[7] https://www.google.com/intl/es/chrome/
[8] https://www.mozilla.org/es-ES/firefox/
[9] https://www.microsoft.com/es-es/windows/microsoft-edge
[10] https://www.opera.com/es
[11] https://www.chromium.org/
[12] https://www.postgresql.org

- Alta concorrencia (*MVCC*): permite acceder a táboas que están sendo escritas en outros procesos sen bloqueos e con consistencia. Isto permite ademais copias de seguridade en quente.
- Soporte nativo para distintos tipos de datos: monetarios, datas, datos sobre redes, etc., así como a creación de tipos propios.
- *Hot Standby*: permite realizar buscas na BD mentres está recuperándose dun erro.
- Notificacións en tempo real.
- Rexistro e gardado de transaccións: trátase dun *log* que permite restaurar a BD a un estado anterior.
- É unha base de datos 100% ACID.
- Herdanza entre táboas.
- Bo soporte da internacionalización.
- Acceso encriptado por *SSL*.
- Estabilidade e fiabilidade: non se producen caídas na base de datos.
- Escalabilidade: *PostgreSQL* pode ser configurado en base ao *hardware* do noso sistema, axustándose ao número de núcleos da *CPU* e á memoria *RAM* dispoñible.
- Permite a autenticación mediante o usuario do sistema operativo.

No momento de decidir a base de datos a utilizar, pensouse nun comezo en *MySQL* dado que era unha BD xa utilizada anteriormente con bo resultado, pero finalmente decantouse a balanza por *PosgreSQL* buscando unha pequena saída da zona de confort, ademais de algunhas considerables vantaxes sobre *MySQL*:

- Cumprimento de ACID en todas as versións.
- Soporte para extensibilidade: *PostgreSQL* permite ser ampliado ou personalizado, mentres que *MySQL* non o soporta.
- Seguridade: cifrado total por *SSL* para *PostgreSQL*, mentres que *MySQL* só o soporta en certas versións.
- Rendemento: *MySQL* é lixeiramente máis rápido en consultas simples, pero en consultas complexas *PostgreSQL* proporciona un maior rendemento.

Emprego no proxecto

Como xa mencionamos anteriormente, unha das opcións máis interesantes desta BD é a posibilidade de autenticarse directamente co teu usuario do sistema. Porén, como se trata dunha aplicación que se debería poder utilizar en diversos equipos con pouca configuración, decidiuse que sería máis sinxelo utilizar a maneira tradicional (un usuario *"videosearch"* que ten as súas bases de datos para uso normal e para test, cada unha coas táboas necesarias).

Para iso, simplemente entramos na aplicación e creamos os usuarios, as bases de datos necesarias e garantimos os permisos.

```
1  postgres=# create user videosearch with password 'root';
2  CREATE ROLE
3  postgres=# create database videosearchdb;
4  CREATE DATABASE
5  postgres=# grant all privileges on database videosearchdb to
   videosearch
6  postgres-# ;
7  GRANT
8  postgres=# create database videosearchdbtest;
9  CREATE DATABASE
10 postgres=# grant all privileges on database videosearchdbtest
   to videosearch;
11 GRANT
12
```

Listing 4.1: Proceso de creación das bases de datos en *PostgreSQL*

A configuración para os test non foi de gran utilidade, xa que debido á utilización de *Jenkins*, era necesario adaptarse ao entorno no que se executaban as probas. En dita máquina non se dispoñía dunha instalación de *PostgreSQL*, polo que se decidiu utilizar *HSQLDB* para a realización dos test, e *PostgreSQL* para a execución da aplicación.

4.2.2 PgAdmin4

PgAdmin4 [13] é unha aplicación web que nos proporciona un entorno gráfico para xestionar bases de datos de *PostgreSQL*, así como manipular e consultar a información. É unha ferramenta de código aberto que, nesta versión, está desenvolvida en *Python*. Elixiuse fronte a outras opcións debido á gran comunidade que existe, o que permite encontrar solución a posibles problemas de maneira sinxela.

[13] https://www.pgadmin.org

4.2.3 HSQLDB

HSQLDB[14] ou *HyperSQL* é unha base de datos desenvolvida en *Java* que permite executarse sen necesidade de instalación. É distribuído mediante licenza *BSD*. Entre as súas principais características están:

- Posibilidade de utilización da memoria *RAM* para a realización de operacións, o que permite un bo rendemento.
- Soporte para illamento de transaccións.
- Sen limitacións de tamaños para operar, unicamente as limitacións de memoria do *hardware*.
- Boa recuperación ante fallos mediante un *redo log*.
- Completamente multifío.
- Soporte para *MVCC*.

Emprego no proxecto

Neste caso foi necesaria a súa implementación como base de datos para a execución dos test da aplicación en *Jenkins*, dado que a máquina na que se executaban ditos tests non posuía unha instalación de *PostgreSQL*. Con este problema, había dúas maneiras de solucionalo:

1. Crear un perfil distinto de execución para cada BD, o cal poderíamos invocar mediante argumentos en liña de comandos con *Maven*.
2. Configurar a execución da aplicación con *PostgreSQL* e a execución dos tests con *HSQLDB*.

Resultou ser elixida a segunda opción xa que permitía unha configuración máis sinxela.

4.2.4 JPA e Hibernate

JPA é a *API* de persistencia estándar para Java, é dicir, un mapeador obxecto-relacional (*ORM*). Pode utilizarse por si soa ou con algunha das súas implementacións, entre as que destaca *Hibernate* [15].

JPA mapea as clases de *Java* ás táboas da BD, simplificando enormemente a programación respecto ao tradicional *JDBC*, xa que ofrece un paradigma 100% orientado a obxectos, o que

[14] http://hsqldb.org/
[15] https://hibernate.org/

permite un desenvolvemento e un mantemento do software máis rápido. Pola contra, é máis lento e pode limitar as funcionalidades da BD respecto a *JDBC*.

Como se indicaba antes, unha das implementacións de *JPA* é *Hibernate*. Trátase dun *framework* que engade numerosas funcións a maiores sobre a especificación, entre elas a linguaxe de consultas orientada a obxectos *HQL*, máis opcións para controlar a caché ou os bloqueos das transaccións a BD, etc. Dado o seu coñecemento previo por parte do alumno, optouse por elixilo como o *ORM* para este proxecto.

Clave primaria

Na creación das claves primarias das entidades, utilizouse a estratexia SEQUENCE. Dita estratexia resulta ser a máis eficiente, xa que permite *Optimistic Locking* e, ademais, a creación de secuencias para a clave primaria está altamente optimizada en *PostgreSQL*. Debido a isto, foi elixida fronte a outras estratexias como TABLE ou IDENTITY, as cales presentaban inconvenientes en tema de rendemento. [33]

4.3 Spring Framework

Spring [16] é un *framework* distribuído baixo a *Apache License* para o desenvolvemento de aplicacións web en *Java*.

Entre as súas características principais están:

- Xestión de transaccións: unifica diversas transaccións de *APIs* e coordínaas para os obxectos *Java*, sen estar ligado ao entorno.
- Contedor de inversión de control (*IoC*): permite a inxección de dependencias e a configuración dos compoñentes da aplicación.
- Lixeiro tendo en conta a súa funcionalidade, en parte debido á súa implementación *POJO*, polo que non vai herdar nin implementar o que non se necesite.
- Programación orientada a aspectos (*AOP*): separa partes recorrentes como por exemplo o login da capa lóxica da aplicación.
- Integración sinxela con outros *frameworks*, como *JDBC*, *Hibernate* e *JPA*.

Conta con diversos submódulos, os cales poden utilizarse independentemente do resto. No proxecto utilizáronse os seguintes:

[16] https://spring.io/

- *Spring Boot* [17]: simplifica a configuración das dependencias e o despregue da aplicación web. En este caso empregouse con *Tomcat*[18].

- *Spring Security* [19]: proporciona ferramentas personalizables para a autenticación dos usuarios, así como o control de acceso en base a permisos.

- *Spring MVC* [20]: facilita a utilización do modelo MVC, o que permite separar a interface gráfica da xestión dos datos e da lóxica de negocio.

Neste caso decidiuse a súa utilización debido ao coñecemento previo por parte do alumno de *Spring MVC* e *Spring Security*, ademais do interese por probar *Spring Boot*

4.4 Aplicación web

Nesta sección descríbense as tecnoloxías utilizadas na capa de presentación.

4.4.1 HTML5

HTML é unha linguaxe utilizada para estruturar e presentar o contido da web, sendo a linguaxe web máis importante. As normas para HTML5[21] son elaboradas polo organismo *W3C*.

Entre as súas melloras respecto a versións anteriores está a inclusión da etiqueta `<video>`, que se utiliza no proxecto e permite engadir vídeos de maneira rápida e sinxela, contando con características como *buffering*, *PiP*, posibilidade de descarga, control de volume ou xanela completa. Para controlar o *buffering* existen varias posibilidades mediante o atributo `preload`, en función das necesidades do usuario [34]:

- `preload="metadata"`: o autor non espera que o usuario necesite o recurso, pero é razoable descargar certas cousas, tales como dimensións, primeiro *frame*, duración, etc. Esta opción parécenos a máis axeitada para as listaxes de vídeos, xa que impide a sobrecarga da rede facendo *buffering* de varios vídeos, pero manténdoos preparados para iniciarse.

- `preload="auto"`: permite facer *buffering*, xa que se considera que non é un risco para o servidor. É a opción utilizada ao mostrar un só vídeo na aplicación.

[17] https://spring.io/projects/spring-boot
[18] http://tomcat.apache.org/
[19] https://spring.io/projects/spring-security
[20] https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html
[21] https://html.spec.whatwg.org/multipage/

- `preload="none"`: o autor non espera que o usuario necesite o vídeo, ou pode ser desexable minimizar o tráfico de rede. Con esta opción non se descarga ningún dato.

4.4.2 CSS3

CSS3 é a ultima versión do estándar *CSS* [22]. É unha linguaxe de deseño gráfico para definir e crear a presentación dun documento escrito nunha linguaxe de marcado, como por exemplo *HTML*. *CSS* está pensado para separar o deseño do documento do seu contido, permitindo maior flexibilidade e control do desenvolvemento.

4.4.3 JavaScript e JQuery

JavaScript é unha linguaxe de programación interpretada, que se utiliza para crear páxinas web dinámicas. Principalmente é empregado no lado do cliente, e permite modificar os elementos das páxinas *HTML*. Isto implica que un usuario pode ver o código e incluso modificalo, polo que se deben realizar comprobacións no servidor de que os datos son correctos e non sufriron modificacións, sobre todo ao utilizar peticións *AJAX*.

Por outro lado, *jQuery* [23] é unha librería *JavaScript* multiplataforma. Ten como obxectivo facer moito máis simple a manipulación de documentos *HTML*, a xestión de eventos, animacións, etc.

4.4.4 Materialize

Materialize [24] é un *framework* para axilizar a creación de interfaces atractivas baseado nas pautas de *Material Design* [25]. Posúe numerosas clases *CSS* para engadir estilos aos elementos *HTML* de forma sinxela e unificada. En conxunto con funcións de *JavaScript*, permite utilizar compoñentes ou estilos que serían demasiado custosos de desenvolver para cada aplicación. Por outro lado, axuda a que a páxina web sexa *responsive*.

Pese a que o alumno non coñecía este *framework*, foi elixido para darlle un deseño distinto ao que se conseguiría con *Bootstrap* [26], co que xa se tiña bastante experiencia.

[22] https://www.w3.org/blog/CSS/
[23] https://jquery.com/
[24] https://materializecss.com/
[25] https://material.io/design/
[26] https://getbootstrap.com/

4.4.5 Thymeleaf

Thymeleaf [27] é un motor de padróns para o desenvolvemento de *XML*/*XHTML*/*HTML5* con *Java*. É executado no lado do servidor e non está limitado a ámbitos web. Posúe módulos para integrarse perfectamente con *Spring*.

Foi elixido pola súa flexibilidade, facilidade de uso e anteriores experiencias satisfactorias do alumno con dita tecnoloxía.

4.5 Lucene

Lucene[28] é unha librería de código aberto para a creación de índices invertidos de texto e a súa posterior busca neles. Está escrito en *Java*, polo que permite ser utilizado en diversas plataformas, aínda que actualmente existen versións para outras linguaxes, como *python* ou *C*. Está distribuído pola *Apache Software Fundation*, a cal tamén ofrece o servidor de buscas *Solr*[29], baseado en *Lucene*. Dado o descoñecemento do alumno neste ámbito, foi elixido por recomendación dos titores, como base para a introdución nos índices invertidos.

Entre as características destacadas de Lucene están:

- Busca de texto completo: é un programa que busca nun conxunto de documentos o termo ou termos desexados polo usuario.
- Indexación incremental: permite actualizar un índice engadindo ou eliminando información sen a necesidade de volver a crear o índice completo.
- Eficiencia: permite procesar datos rapidamente cunhas necesidades de *RAM* moi baixas. O almacenamento do índice tamén está optimizado.
- Busca por puntuación: Os resultados son devoltos en orde segundo a súa puntuación na busca.
- Buscas complexas: *Lucene* permite realizar consultas de varios termos, con expresións regulares, con operadores booleanos, con comodíns, etc.

A arquitectura de Lucene estruturase desta maneira:

[27] https://www.thymeleaf.org/
[28] https://lucene.apache.org/core/
[29] https://lucene.apache.org/solr/

- Directorio (*Directory*): é unha lista de arquivos sen xerarquía, é dicir, non hai subcarpetas. Permite abstraerse de como ditos arquivos son gardados.
- Documento (*Document*): é a unidade en base á cal se compoñen os índices, e á súa vez está composto por *Fields*. Os documentos permiten indexar e buscar, e son gardados nun *Directory*.
- Campo (*Field*): un campo ou *field* é unha tupla [título:valor].
- Termo (*Term*): o valor de cada *field* está composto por unha secuencia de termos. [35] [36]

Emprego no proxecto

No caso desta aplicación, o tratamento de datos con Lucene pódese dividir nas seguintes fases:

1. Análise: en esta fase necesitamos eliminar as *stopwords*[30] e facer *stemming*. Para iso, utilizamos un analizador adaptado ao idioma galego, o `GalicianAnalyzer` [31].
2. Indexación: a partir dos datos en bruto contidos no arquivo de texto proporcionado pola transcrición de *Kaldi* (exemplo na figura 4.3), creamos os *fields* cos seus respectivos títulos e termos. A transcrición está dividida en frases. Para cada frase de cada vídeo, contamos coas dez mellores transcricións e gardámolas cada unha nunha entrada. O título do *field* corresponde co id en base de datos do vídeo correspondente á dita entrada, mentres que os termos son os resultados da transcrición, é dicir, unha frase. Ditos *fields* son creados coa opción `Field.Store.YES`, o que indica que os resultados serán gardados no índice e poderán ser buscados. Para que isto ocorra, necesitamos enviarlle os documentos nos que están gardados os campos anteriores a un `IndexWriter`, que é o indexador de Lucene. Esta clase, a través da súa configuración, obtén unha instancia do analizador escollido e realiza a análise do que falábamos na fase anterior para posteriormente indexar os datos e almacenalos no directorio escollido.
3. Busca: para as consultas necesitamos coñecer o título do *field* e as palabras a buscar. Con isto, e axudándonos de novo do `GalicianAnalyzer`, obtemos os resultados con maior puntuación. A partir deles, podemos obter os *Terms* dentro de cada campo. Dado que estamos almacenando os 10 mellores resultados proporcionados por Kal-

[30] Lista de *stopwords* de galego https://github.com/apache/lucene-solr/blob/master/lucene/analysis/common/src/resources/org/apache/lucene/analysis/gl/stopwords.txt

[31] https://lucene.apache.org/core/7_5_0/analyzers-common/org/apache/lucene/analysis/gl/GalicianAnalyzer.html

di, algúns son moi semellantes entre si, pero entre todos axudan a obter unha mellor transcrición final.

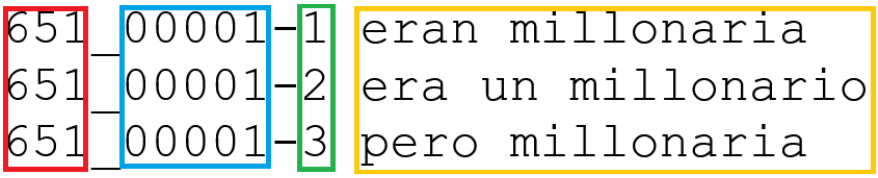


Figura 4.3: Exemplo de tres entradas no arquivo de saída do recoñecedor de voz. En vermello móstrase o id en base de datos do vídeo utilizado. En azul, número do fragmento no que se dividiu a transcrición. O cadrado verde representa a clasificación das transcricións máis axeitadas. Por último, o cadrado laranxa mostra o texto transcrito.

4.6 FFmpeg

FFmpeg [32] é unha colección de software libre coa que podemos realizar conversións, streaming ou gravacións de audio e vídeo. Foi escollido debido á posibilidade de crear *scripts* que o chamen e á gran comunidade que existe, o que permite resolver dúbidas facilmente.

Neste proxecto é utilizado para extraer en formato *wav*, o audio de vídeos en formato *avi*, que á súa vez son transformados posteriormente a formato *mp4* para permitir a súa reprodución. No código inferior podemos observar o *codec* e os atributos necesarios para o proceso de extraer o audio do vídeo, cunha única canle de audio e unha frecuencia de 16000Hz [37], mentres se mantén o nome do ficheiro e se lle cambia a extensión.

```
1  for vid in ./videos/not_processed/*.avi;
2  do ffmpeg -i "$vid" -vn -acodec pcm_s16le -ac 1 -ar 16000
   "${vid%.avi}.wav";
3  mv videos/not_processed/*.wav ./audiosgalego
4  done
5  mv videos/not_processed/*.avi ./videos
6
```

Listing 4.2: Transformación dun arquivo de vídeo en formato *avi* a un arquivo de audio *wav* con unha canle de son e unha frecuencia de 16kHz

[32] https://ffmpeg.org/

4.7 Ferramentas de soporte

4.7.1 Git

Git [33] é un sistema de control de versións deseñado para manexar con velocidade e eficiencia proxectos de todos os tamaños. É distribuído con licenza de código aberto. Algunhas das súas características máis destacadas son [38]:

- Forte apoio ao desenvolvemento non lineal: permite xestionar rapidamente as ramas e as distintas versións. Especialmente útil cando os proxectos contan con varios membros traballando en paralelo.
- Xestión distribuída: cada programador ten no seu equipo unha copia enteira do proxecto.
- Sistema de ramas flexible: é posible crear ramas locais independentes dun repositorio centralizado. Tamén se poden crear ramas con diversos propósitos para despois unir os cambios.
- Xestión eficiente de proxectos grandes debido á eficiencia da xestión de diferencias entre arquivos.

Cos motivos indicados e sendo practicamente o estándar da industria para novos proxectos, a elección é bastante sinxela, respecto a outras alternativas como *SVN*.

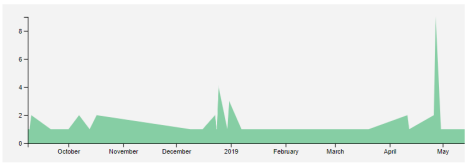
GitLab

GitLab [34] é un servizo web de control de versións baseado en *Git*, que conta cunha versión libre e gratuíta e varias versións de pago con funcionalidades máis avanzadas. Permite, entre outras cousas:

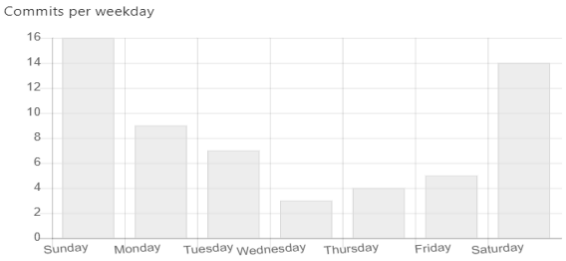
- Revisión do código.
- Xestión de *merge requests*.
- Configuración e execución automática de *pipelines*.
- Seccións para *issues* e *to-dos*.
- Opcións de configuración para traballar en conxunto con ferramentas de *CI*/*CD* de terceiros.

[33] https://git-scm.com/
[34] https://about.gitlab.com/

- Métricas do proxecto.
- Creación de *wikis*.



(a) Historial de *commits* á rama *dev*, excluíndo *merges*



(b) *Commits* segundo o día da semana

Figura 4.4: Exemplo de gráficas proporcionadas por GitLab

4.7.2 Maven

Maven [35] é unha ferramenta de código aberto para a construción de proxectos *Java* e a súa xestión. Proporciona unha gran simplificación e estandarización do proceso de *build* do código, permitindo compilar calquera proxecto da mesma maneira e en todas as etapas, dende a configuración inicial até o despregue. Para describir o proxecto e as súas dependencias, utiliza un arquivo *xml* de configuración coñecido como *POM*. Entre as súas características destacadas están:

- Creación sinxela e áxil dun novo proxecto ou módulo.
- Xestión das dependencias propias e de terceiros definidas no `pom.xml`, incluíndo descargas dende a rede e actualizacións automáticas. *Maven* anima a empregar repositorios centrais de dependencias.
- Posibilidade de traballar con múltiples proxectos á vez.
- Estandarización da estrutura dos proxectos: código fonte, recursos, arquivos de configuración, etc.
- Extensible: permite a creación ou utilización de *plugins*.

Decidiuse a utilización de *Maven* fronte a outras opcións debido á experiencia no seu uso do alumno.

[35] https://maven.apache.org/

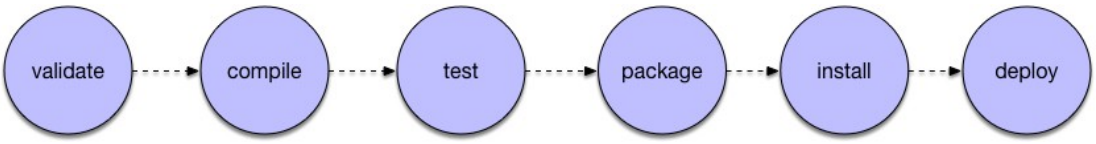


Figura 4.5: Ciclo de vida de *Maven*. Se precisamos executar unha fase en concreto, realizarase despois de todas as anteriores. [13]

4.7.3 Jenkins

Jenkins [36] é un servidor de automatización de código aberto escrito en *Java*. Preséntase como unha web e é utilizado para a integración continua (*CI*), é dicir, construír e testar os proxectos continuamente. Tamén é utilizado para proporcionar entrega continua (*CD*). Estas funcionalidades permiten facilitar e minimizar o esforzo requirido para obter unha nova versión. Por outra parte, tamén permite integrar outras tecnoloxías de test e de despregue.

Un correcto uso da *CI* permite coñecer rapidamente o estado do *software*, detectar os fallos antes e reducir as tarefas repetitivas e manuais.

Debido aos beneficios que proporciona e ao previo coñecemento deste software, decidiuse a súa utilización fronte a outras alternativas.

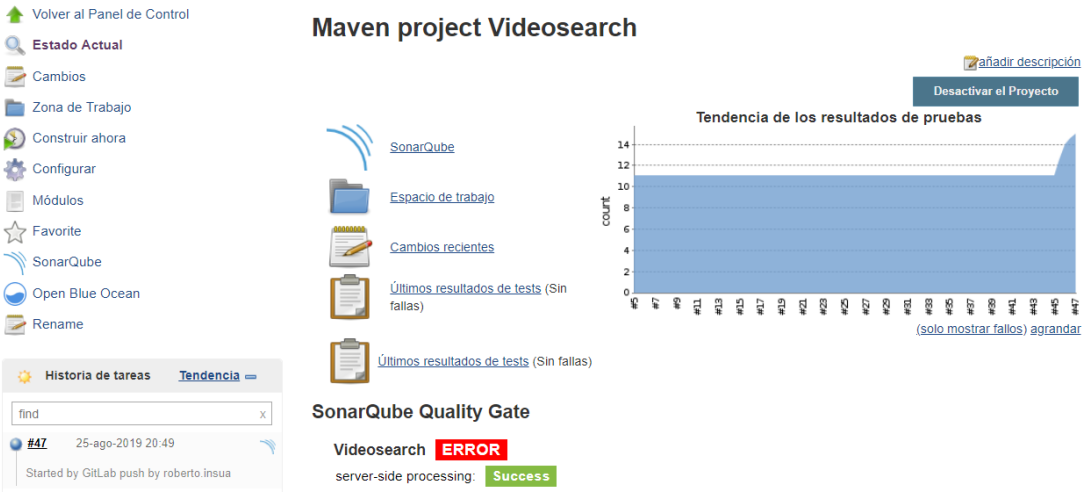


Figura 4.6: Exemplo de integración continua ao longo do desenvolvemento do proxecto

Emprego no proxecto e configuración

Para o emprego de *Jenkins*, compre establecer unha comunicación entre este e o repositorio, neste caso *GitLab*. Para iso, débese engadir na configuración de *Jenkins* a *URL* do

[36] https://jenkins.io/

repositorio xunto coas credenciais. Isto permitiríanos facer tests manualmente. Como o obxectivo é que se realicen automaticamente, é necesario configurar en *Gitlab* un *Webhook* ou chamada HTTP, que avise ao servidor de *Jenkins* cando ocorra algo, no noso caso, cando se realice un *push*.

Ademais, debemos elixir as ramas ás que lle queremos facer tests. Non ten sentido facer tests para cada *commit* en ramas con funcionalidades sen completar, xa que é moi posible que fallen, pero nun momento no que é lóxico e asumible. Por ese motivo, engadimos para que se fagan test só das ramas *master*, *dev*, *hotfix* e *hsqldb*, xa que esta última contiña cambios de configuración que implicaban a *Jenkins*.

4.7.4 SonarQube

SonarQube [37] é unha plataforma *Open Source* de inspección continua, a cal consiste en medir a calidade interna (robustez, complexidade, facilidade de mantemento...) do proxecto *software*. Para iso, realiza unha análise estática do código fonte en busca de erros ou *code smells* coa intención de mellorar a calidade do produto.

En base a esa avaliación, infórmanos sobre duplicidades do código, cobertura dos test, vulnerabilidades, complexidade ciclomática, estándares de codificación non cumpridos, etc., en base aos cales, podemos cumprir ou non o *Quality Gate*.

Debido aos beneficios que proporciona e ao previo coñecemento deste software, decidiuse a súa utilización fronte a outras alternativas.

Configuración

Para dispoñer das vantaxes de *SonarQube*, foi necesaria realizar a seguinte configuración:

- Engadir o *plugin* de *Jacoco* [38] para *SonarQube*, necesario para contar coas probas de cobertura.
- Definir nos *goals* de execución en *Jenkins* os parámetros necesarios para limpar o directorio, invocar a *Jacoco* e realizar un *mvn install*.
- Engadir un *paso posterior* en *Jenkins* que se realiza só cando a execución foi exitosa.

  Configurar as propiedades do análise para enviar a *SonarQube* a información necesaria para realizar os seus informes.

  Engadir argumentos adicionais aos *goals* para crear o proxecto en *SonarQube* coa información das ramas.

[37] https://www.sonarqube.org/
[38] https://www.eclemma.org/jacoco/

4.7.5 Eclipse Photon

Eclipse[39] é un *IDE* ou entorno de desenvolvemento integrado. Soporta múltiples linguaxes e proporciona un espazo de traballo amplamente personalizable e adaptado ás necesidades do proxecto. *Eclipse Photon* é a ultima versión do *IDE*. Foi escollido debido ao coñecemento previo por parte do alumno e ao seu correcto funcionamento, fronte a experiencias previas non moi satisfactorias con outros *IDE*.

Entre as características máis destacadas están:

- Gran colección de *plugins*: é posible aumentar a funcionalidade da ferramenta mediante os *plugins*, tanto publicados por *Eclipse* como por terceiros.
- Perspectivas: o concepto de traballo está baseado nas perspectivas, que son unha preconfiguración de xanelas e editores, que se modifican segundo o traballo que esteas realizando nese momento.
- Depurador: conta tamén cunha perspectiva específica e permite comprender mellor o código, sobre todo cando buscamos algún erro.
- *Code completion*: permite completar o código automaticamente con suxestións dependentes do contexto.
- *Save Actions*: na configuración de *Eclipse* existen opcións que nos permiten aforrar tempo nas tarefas repetitivas. Neste desenvolvemento utilizáronse entre outras dar formato automático ao código, a organización dos *imports*, a xestión das anotacións ou a eliminación dos *cast* innecesarios. Estas accións execútanse ao gardar o arquivo.
- Integración con outro software: permite utilizarse conxuntamente con sistemas de integración continua, xestión de tarefas, etc.
- Altamente personalizable: pode ser configurado ao teu gusto, grazas aos *plugins* como á súa ampla configuración.

4.7.6 JUnit4

JUnit [40] é un *framework* para crear probas repetibles en *Java*. Isto permite crear probas unitarias para cada método e comprobar que se comporta como se espera, xa que permite comprobar os valores devoltos xunto con outros definidos polo usuario como agardados mediante aserción. Ademais permite que cada método se desenvolva nun entorno estándar e, que ao

[39] https://www.eclipse.org/
[40] https://junit.org/

finalizar, os cambios sexan revertidos para que a seguinte proba volva a ter o mesmo entorno estándar que antes. Foi escollido debido ao correcto funcionamento que experimentou o alumno en experiencias previas.

Test-driven development

O desenvolvemento guiado por probas ou *TDD* é un proceso para desenvolver o software que consiste en escribir as probas antes de desenvolver a funcionalidade. Desta maneira, conseguimos que o funcionamento se acople ao test, que estará feito seguindo as necesidades da nova funcionalidade, en vez de que o test se acople aos resultados do código realizado. Este proceso foi utilizado en algunhas funcionalidades con bo resultado.

4.7.7 Tomcat

Tomcat [41] é un servidor web de código aberto que implementa *Java Servlet*, *JavaServer Pages*, *Java Expression Language* e *Java WebSocket*.

Debido á súa popularidade, conta cunha ampla comunidade e documentación. Iso sumado ao seu coñecemento previo por parte do alumno, motivou o seu uso.

4.7.8 MagicDraw

MagicDraw [42] é unha ferramenta utilizada para a creación dos diagramas UML. É de pago, pero decidiuse o seu uso dado que o alumno contaba cunha licenza para uso académico e experiencia previa ca ferramenta, ademais de que é unha solución máis completa que outras coas que xa se tiña experiencia, como *yUML* [43] ou *Dia* [44].

[41] http://tomcat.apache.org/
[42] https://www.nomagic.com/products/magicdraw
[43] http://yuml.me/
[44] http://dia-installer.de/

Capítulo 5

Proceso de enxeñería

5.1 Elección da metodoloxía

Unha correcta metodoloxía pode supoñer a diferenza entre un proxecto fracasado e un exitoso, polo que é necesaria unha elección meditada e un correcto seguimento. Debido á experiencia en traballos pasados, optouse por unha metodoloxía áxil.

As metodoloxías áxiles proporcionan rapidez e flexibilidade definindo as preferencias sobre o que se debe buscar durante o desenvolvemento:

- Individuos e interaccións sobre procesos e ferramentas.
- *Software* que funciona sobre documentación exhaustiva.
- Colaboración co cliente sobre negociación de contratos.
- Responder ante o cambio sobre seguimento dun plan.

Entre as distintas metodoloxías áxiles, elixiuse *Scrum* [1], debido á súa contrastada eficacia e a que consegue ser áxil mentres se mantén a orde sobre o que se está realizando cuns procesos ben definidos. Porén, a filosofía de *Scrum* é a de ser un *framework* flexible que proporcione as ferramentas necesarias para conseguir que os proxectos sexan exitosos, non unha metodoloxía á que se deba seguir cegamente. Grazas a isto, foi posible adaptar a metodoloxía ás necesidades do proxecto.

5.2 Scrum

Scrum é un marco de traballo que define un conxunto de prácticas e roles, cos que se definirá o proceso de desenvolvemento utilizado. Divídese en iteracións, coñecidas como

[1] https://www.scrum.org

sprints de entre 2-4 semanas de duración. Dita duración pode ser modificada, pero debe ser constante para que o equipo se acostume e poida render adecuadamente. Durante ditos *sprints* están prohibidos os cambios nas prioridades para proporcionar certa estabilidade.

Un dos aspectos clave de *Scrum* é a *Definition of Done* (*DoD*). Implica definir entre os membros cando se pode dicir que un traballo está feito e finalizado. Unha funcionalidade, para ter o cualificativo de *done* debería estar integrado, probado, cumprir con certos criterios de calidade, etc. En definitiva, debe ser un produto potencialmente entregable e utilizable.

Roles

Os principais roles utilizados en *Scrum* son:

- *Scrum Master*: trátase dun xestor e desenvolvedor a partes iguais. Busca o correcto funcionamento do equipo e elimina impedimentos que poidan xurdir tomando decisións rápidas. É o responsable de promover *Scrum* e de actuar de representante do equipo ante a dirección e viceversa. Tamén debe actuar de escudo do equipo ante interferencias externas.
- *Product owner*: é a persoa que representa os intereses do cliente. Define as funcionalidades do produto software e prioriza as que se desenvolverán antes e as que se entregarán en cada *sprint*.
- *Team*: É un grupo de (idealmente) entre 5-9 persoas de diversos perfiles. Debe organizarse por si mesmo.

Reunións

As reunións establecidas por *Scrum* son as seguintes:

- *Sprint Planning meeting*: o equipo selecciona de entre as funcionalidades e requisitos pendentes, aqueles que pode realizar durante esta iteración, e crea con eles a Axenda do Sprint. En dita axenda, identifícanse tarefas e realizase a estimación.
- *Daily Scrum Meeting*: pequena reunión diaria para identificar problemas e na que só pode falar o equipo, *Scrum Master* e o *Product Owner*. Nela contéstase a tres preguntas:

  Que fixeches onte?

  Que vas facer hoxe?

  Hai obstáculos no teu camiño?

- *Sprint Review*: reunión interna e informal do equipo no que presenta o realizado durante ese *sprint*. Todo o mundo está invitado.

- *Sprint Retrospective*: tras cada sprint, bótase un ollo ao que funciona e ao que non entre todos os roles. Unha maneira de realizar unha retrospectiva sería discutir sobre o que lles gustaría:

  Comezar a facer.

  Deixar de facer.

  Continuar facendo.

Artefactos

Scrum conta con diversos artefactos, os cales xestionaremos con Taiga:

- *Product Backlog* ou Axenda de Produto: é unha lista priorizada de todos os requisitos do proxecto. Cada requisito debe ter valor para o cliente ou para os usuarios. É priorizada polo *Product Owner* e repriorizada ao comezo de cada *sprint*.
- *Sprint Backlog* ou Axenda de Sprint: define as tarefas a realizar durante o *sprint* que foron seleccionadas no *Sprint Planning*. Cada tarefa conta cunha estimación do traballo restante que se debe actualizar a diario. Cada membro do equipo escolle as tarefas a realizar.
- *Sprint Burndown*: trátase un gráfico do traballo pendente e completado ao longo do *sprint*.
- Velocity: é a media do traballo completado por *sprint*. Axuda a mellor as estimacións en futuros *sprints*.

Adaptación ao proxecto

A adaptación ao proxecto realizouse da seguinte maneira:

- O rol de *Product Owner* foi desempeñado polos directores do proxecto, mentres que o de *Scrum Master* e o de *Team* foi exercido polo alumno.
- A duración dos sprints foi establecida como de dúas semanas.
- Realizouse un *Product Backlog* cos requisitos tras un par de reunións co *Product Owner*. Como en este caso as reunións eran entre persoas do gremio, os requisitos poden conter algún tecnicismo máis dos recomendados.
- Antes de cada *sprint*, realízase un pequeno *Sprint Planning* para decidir o que se debería facer no seguinte sprint.

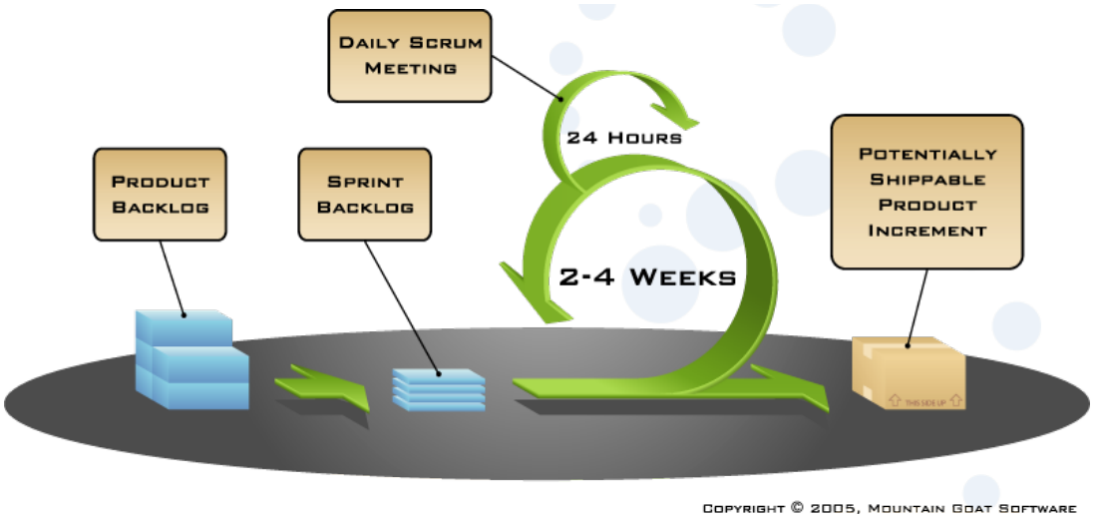


Figura 5.1: Proceso de *Scrum*

- Tras cada *sprint*, realízase unha pequena *Sprint Review* e coméntase o avanzado neste último período.

- O maior problema encontrado foi relativo á duración e ao período entre sprints. Scrum anima a definir o tempo máximo para conseguir certos obxectivos, unha técnica coñecida como *timeboxing*. Debido tanto a motivos laborais como persoais, non sempre era posible realizar os sprints no período de dúas semanas, ou incluso realizar varios sprints de maneira consecutiva e sen paradas. Por este motivo, non se puideron cumprir os elementos do *timeboxing*. Porén, intentouse axustar o esforzo realizado en cada *sprint* para que fose sempre o mesmo.

Excepcións respecto a Scrum

Os cambios respecto a *Scrum* na realización deste proxecto foron os seguintes:

- *Reunións*: dado que o proxecto era realizado por só unha persoa, e debido a que por motivos laborais e de estudos non era posible unha constancia diaria no traballo do sprint, suprimiuse a realización do *Daily Scrum Meeting*.

- *Sprint Backlog*: a inclusión de tarefas para cada *sprint* foi complicada, xa que hai historias de usuario definidas no *Product Backlog* dificilmente divisibles e que poderían necesitar por si mesmas varias iteracións. Ditas historias, dende un punto de vista técnico si que son divisibles, polo que se optou por que o *Sprint Backlog* non tivese unha correspondencia exacta coas historias de usuario do *Product Backlog*.

- *Incrementos de produto*: debido ás tarefas definidas no punto anterior, non sempre foi posible realizar un incremento de produto ao final de cada sprint. Tras asentar as bases coas tecnoloxías menos coñecidas polo alumno, realizáronse correctamente.

5.3 Xestión do proxecto

Unha correcta xestión do proxecto é vital para conseguir os obxectivos marcados, cuns estándares de calidade á altura do esperado e mantendo a raia o tempo empregado e os custos.

A continuación detallase as xestións e planificacións realizadas.

5.3.1 Product Backlog

O *Product Backlog* do proxecto estaba formado polas historias de usuario detalladas a continuación.

1. Como **usuario** quero **subir vídeos** para **engadir contido á web**.
2. Como **usuario** quero **realizar unha busca** para **atopar os vídeos que preciso**.
3. Como **usuario** quero **ver os meus vídeos subidos**.
4. Como **usuario** quero **poder marcar os vídeos como favoritos**.
5. Como **usuario** quero **ver os meus vídeos favoritos**.
6. Como **usuario** quero **ver o meu historial de buscas**.
7. Como **usuario** quero **rexistrarme e iniciar sesión** para **acceder a máis funcionalidades**.
8. Como ***Product Owner*** quero **que a información sobre os vídeos se garde de maneira eficiente** para **permitir un bo rendemento na busca**.
9. Como **usuario** quero **reproducir os vídeos**.
10. Como **usuario** quero **realizar as buscas por voz**.
11. Como **administrador** quero **ter o control do procesado dos vídeos** para **evitar unha sobrecarga do sistema nun momento de alta demanda**.
12. Como **Product Owner** quero **que se garanta a calidade do código**.
13. Como ***Product Owner*** quero **que se transcriban as conversas dos vídeos** para **poder realizar buscas con esa información**.

5.3.2 Tarefas

Como se comentou previamente, debido á complexidade de certas historias de usuario que serían imposibles de rematar nun só *sprint*, estas reconvertéronse a tarefas, nas que xa se tiñan máis en conta os requisitos técnicos, e foron as que se utilizaron na planificación de cada sprint. Son as seguintes:

1. **Transcrición do audio dos vídeos**: primeira toma de contacto co recoñecedor de voz proporcionado. Lectura de documentación e estudo do seu funcionamento. Posteriormente poñerase a proba co procesamento do audio duns vídeos. Consta coas seguintes subtarefas:

   Test e probas con Kaldi.

   Engadir os vídeos necesarios para a súa transcrición.

   Realizar o recoñecemento de voz a texto.

2. **Xestión de arquivos e executables dende *Java***: conseguir executar o recoñecemento de voz dende Java e realizar os tests acordes. Para iso requiría lectura previa de documentación. Consta coas seguintes subtarefas:

   Creación de interface e implementación.

   Tests.

3. **Configuración do almacenamento**: instalación dunha base de datos, que será utilizada no proxecto *Java* mediante un *ORM*. Consta coas seguintes subtarefas:

   Engadir base de datos.

   Configuración da persistencia.

4. **Indexación de palabras**: aos datos obtidos do arquivo de texto xerado polo recoñecedor de voz, realizar unha indexación destes mediante *Lucene*. Requiría lectura previa de documentación.

5. **Procura de texto**: busca de texto no índice invertido de *Lucene* construído na tarefa número 4.

6. **Rexistro de usuarios (modelo)**: implementación e configuración necesaria para o rexistro de usuarios.

7. **Configuración de integración e inspección continua**: configuración e probas para a correcta utilización de *Jenkins* e *SonarQube*. Consta coas seguintes subtarefas:

   Configuración *Jenkins*.

Configuración de *PostgreSQL* e *HSQLDB*.

Configuración de *Jacoco* para *SonarQube*.

8. **Adaptación dos índices á nova saída do recoñecedor**: melloras realizadas no indexado e na procura de texto tras unha modificación do recoñecedor de voz para fragmentar os resultados da transcrición.

9. **Visualización de vídeos**: reproducilos dende a web.

10. **Almacenamento das transcricións**: modificación no sistema de indexado para gardar as transcricións en disco.

11. **Busca por texto dende web**: implementación dun buscador de vídeos dende a web.

12. **Listaxe de resultados básica**: mostrar os resultados devoltos polo buscador.

13. **Rexistro de usuarios (web)**: continuación da tarefa número 6, implementando o acceso e o rexistro na páxina web.

14. **Control de acceso**: configuración das páxinas permitidas para usuarios identificados e para invitados.

15. **Busca por voz dende web**: utilización dalgún tipo de recoñecedor web para poder buscar ditando as palabras. Precisa de investigación para seleccionar a opción máis adecuada.

16. **Vídeos favoritos**: posibilidade de marcar vídeos como favoritos e visualizar posteriormente a lista.

17. **Historial de buscas**: gardado das buscas realizadas polo usuario e creación dunha páxina para a súa visualización.

18. **Subida de vídeos por parte do usuario**: implementación da subida dun vídeo e do seu procesado. Conta con catro subtarefas:

    Subida do vídeo.

    Creación do panel de administración para controlar os procesados.

    Procesado de vídeo polo recoñecedor.

    Obtención de información.

5.3.3 Estimación

Para a estimación do esforzo requirido polas tarefas, utilizouse unha medición con puntos de historia en vez de con horas de traballo. Aínda así, realizouse unha equivalencia inicial orientativa:

$$1punto \approx 2horas$$

Tendo isto en conta, decidiuse realizar sprints de 20 puntos de historia, o que equivale aproximadamente a 40h por sprint, ou 20h semanais.

A estimación inicial foi de 8 sprints, cuns 160 puntos de historia en total. Porén, debido á aparición de novos requisitos, e de certas re estimacións, os puntos definidos finalmente foron 184.

Tarefa	Puntos	Tarefa	Puntos	Tarefa	Puntos
1	20	7	20	13	10
2	20	8	8	14	8
3	10	9	13	15	10
4	5	10	8	16	8
5	10	11	5	17	5
6	8	12	3	18	13

Táboa 5.1: Estimación definitiva das tarefas cun total de 184 puntos de historia

5.3.4 Desenvolvemento do proxecto

Para iniciar este proxecto, decidiuse comezar probando as tecnoloxías que eran novas para o alumno, como o recoñecedor de voz e *Lucene*, para seguir desenvolvendo o resto do proxecto en torno ao aprendido. Debido a iso, e como se pode ver na táboa 5.3, o últimos sprints xa están mais enfocados a crear funcionalidades para a web.

Como se pode observar no gráfico de burndown 5.2, un proxecto planeado para rematar en 8 *sprints*, finalmente levou 11, debido ás xa mencionadas replanificacións e novas historias, ademais do ritmo conseguido polo alumno, o cal foi menor do esperado.

5.4 Custos

Para o cálculo dos custos do proxecto tívose en conta o número de sprints realizados (11) xunto cos puntos de historia teóricos por sprint (20). Para o cálculo do custo dos directores estimouse unha media de 2h en conxunto por *sprint*.

Sprint	Tarefas	Puntos historia
1	1	20
2	2	20
3	3	10
4	4, 5, 6	23
5	7	20
6	8	8
7	9	13
8	10, 11, 12	16
9	13, 14	18
10	15, 16	18
11	17, 18	18

Táboa 5.2: Distribución das tarefas para cada *sprint* e puntos de historia completados en cada un, obtendo unha media de 17.

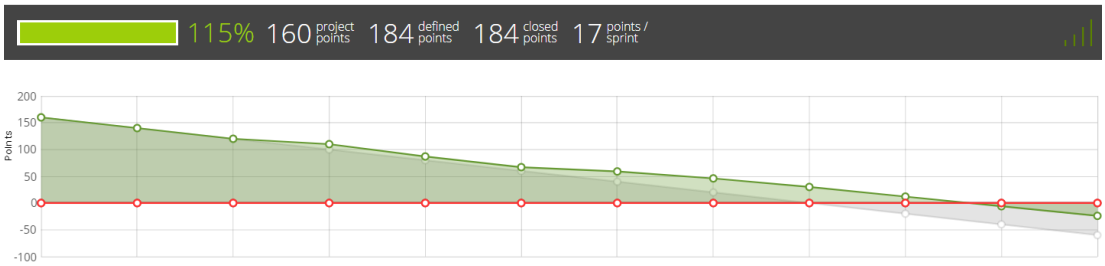


Figura 5.2: Gráfico de *burndown* do proxecto proporcionado por *Taiga*. Podemos ver o efecto dos cambios no proxecto e do ritmo conseguido polo alumno

Os salarios foron extraídos dun estudio salarial do sector TI en Galicia entre os anos 2015-2016 [39]:

Posto	Salario anual bruto	Custo anual empresa	Custo por hora bruto
Programador *Java* Web (Junior)	18000	23580	13,1
Director de informática	38000	49780	27,7

Táboa 5.3: Salarios medios para os participantes do proxecto en euros

Tendo en conta a estimación realizada anteriormente de que:

$$1punto \approx 2horas$$

O custo total do proxecto ascende a 6373 €.

Posto	Horas por sprint	Nº de sprints	Salario por hora bruto	Total (€)
Programador *Java* Web (Junior)	40	11	13,1	5764
Director de informática	2	11	27,7	609
			Total	**6373**

Táboa 5.4: Desglose do custo do proxecto

Capítulo 6

Desenvolvemento

Neste capítulo da memoria detallase o proceso do desenvolvemento do proxecto: requisitos, análise, deseño e implementación.

6.1 Análise de requisitos

Un proxecto de software debe contar cunha correcta análise de requisitos para evitar incorreccións e malentendidos entre o que realmente se pretendía e o resultado final do mesmo.

6.1.1 Requisitos funcionais

Os requisitos funcionais son aqueles que describen calquera función ou comportamento que un software debe realizar. En este proxecto foron identificados os seguintes:

1. Procura de vídeos.
2. Transcrición de vídeos.
3. Rexistro de usuarios.
4. Autenticación de usuarios.
5. Subida de vídeos.
6. Historial de procuras.
7. Marcado e visualización de favoritos.

6.1.2 Requisitos non funcionais

Os requisitos non funcionais especifican criterios a cumprir polo sistema. Non especifican unha funcionalidade, senón características e cualidades necesarios para que o sistema sexa utilizable. Neste proxecto foron identificados os seguintes:

1. Facilidade de uso.
2. Fiabilidade.
3. Mantenibilidade.
4. Rendemento.
5. Seguridade.

6.1.3 Actores e casos de uso

Os usuarios interactuarán de diversas maneiras coa aplicación, dependendo do rol que desempeñen. Neste proxecto existen tres actores:

- Usuario non rexistrado.
- Usuario rexistrado.
- Administrador.

Os casos de uso para cada tipo de actor están detallados na figura 6.1, e relacionados coas historias de usuario na táboa 6.1.

Caso de uso	Historia de usuario
Rexistro	7
Busca de vídeos	2, 10
Reprodución de vídeos	9
Marcar/Desmarcar favoritos	4
Visualización de favoritos	5
Subida de vídeos	1
Visualización de vídeos subidos	3
Visualización de historial	6
Login	7
Iniciar procesado de vídeos	11

Táboa 6.1: Relación entre os casos de uso e as historias de usuario do *Product Backlog* detallado no apartado 5.3.1

6.2 Arquitectura proposta

En base aos requisitos identificados, plantexouse unha arquitectura baseada en capas para a aplicación web. As arquitecturas baseadas en capas permite un desacoplamento das partes

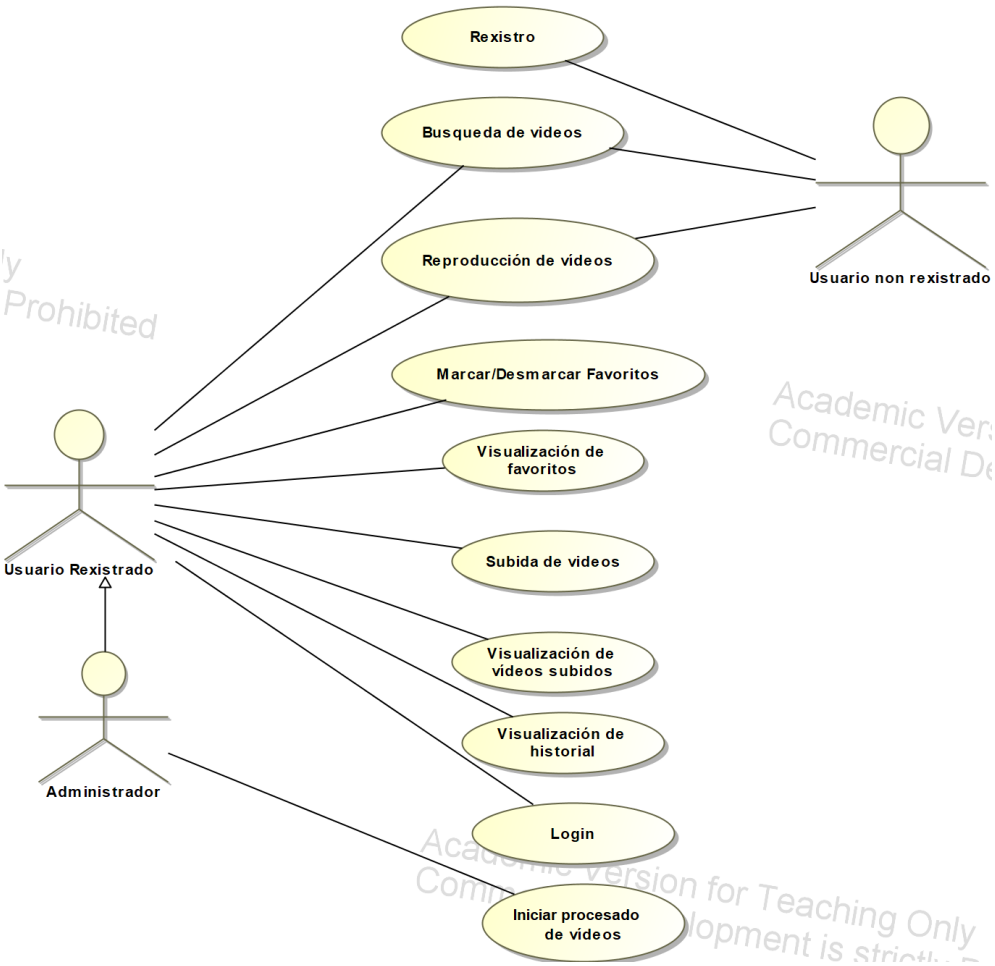


Figura 6.1: Representación dos casos de uso para cada actor.

que compoñen o software, permitindo intercambiar algunha delas no futuro sempre que se cumpran as interfaces previamente definidas. Como se pode observar na figura 6.2 (páxina 52), neste proxecto definíronse tres capas:

- Capa de presentación: trátase da capa coa que interactúa o usuario. Consume a capa de negocio.

- Capa de negocio: contén as regras de negocio da aplicación. Consume a capa de acceso a datos e, neste caso, comunicase co recoñecedor de voz.

- Cama de acceso a datos: accede a base de datos e contén as entidades.

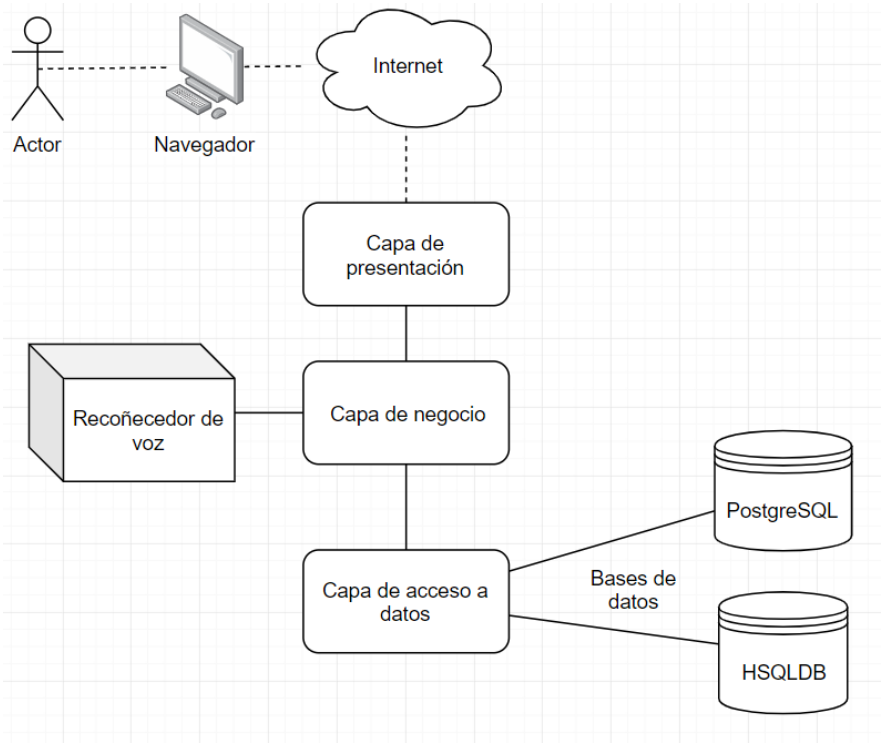


Figura 6.2: Representación da arquitectura do sistema.

6.2.1 Diagrama de clases

A continuación tamén se mostra o modelo de datos 6.3 (páxina 53), o cal mostra as entidades gardadas en base de datos. O diagrama define só os atributos das entidades, xa que as operacións que conteñen eran unicamente *getters* e *setters*. É un modelo de datos sinxelo, dado que a complexidade do proxecto reside no tratamento dos datos.

6.2.2 Servizos

A lóxica da aplicación realízase nos servicios contidos na capa de negocio. Ditos servizos están definidos en interfaces que posteriormente son implementados. Como se pode observar na 6.4 (páxina 57), contamos cos seguintes servicios:

- **KaldiService**: contén funcións relativamente xenéricas para o tratamento de arquivos, pero necesarias para poder executar o recoñecedor dende a aplicación *Java*
- **LuceneService**: permite xestionar a indexación, a busca e a recuperación dos datos dos datos das transcricións realizadas co recoñecedor de voz.
- **SearchService**: define funcións relacionadas coas buscas e recuperacións de vídeos, dende as realizadas coa axuda de *Lucene*, até as máis sinxelas, que só buscan en base de

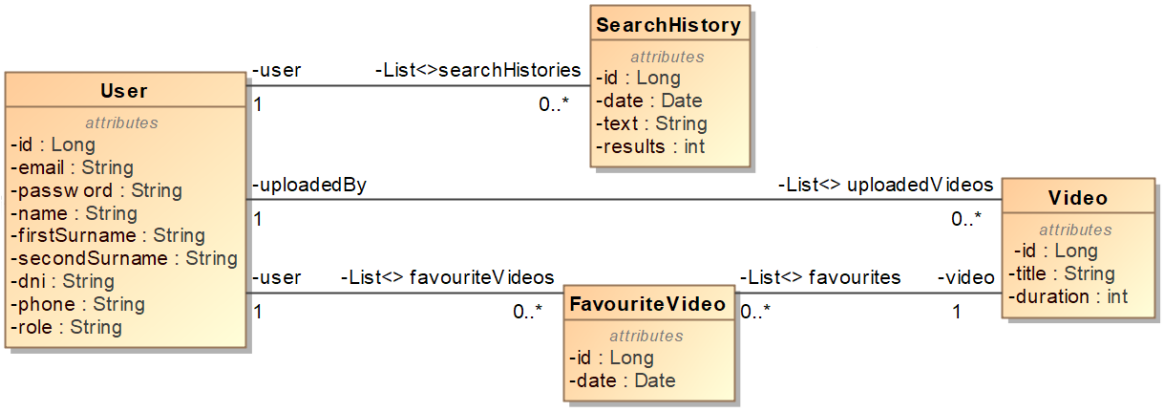


Figura 6.3: Diagrama de clases da aplicación.

datos. Tamén se utiliza para o gardado e recuperación do historial de buscas.

- **VideoService**: é utilizado para a subida de vídeos, así como para a xestión dos vídeos favoritos de cada usuario.
- **UserService**: contén funcións xenéricas sobre a xestión de usuarios.

6.3 Camiño de datos para o procesado dos vídeos

Como se mencionou anteriormente, o máis complexo deste proxecto é o tratamento dos datos para a súa indexación, polo que será explicado a continuación. Debido ás dimensións do diagrama de secuencia, decidiuse dividir en pequenas partes, xunto coas súas explicacións.

1 - Gardado do vídeo

Para que un usuario poida subir os seus vídeos, realízase a seguinte secuencia, a cal se pode observar na figura 6.5 (páxina 58):

- Un usuario rexistrado accede á páxina de subida de vídeo, selecciona o arquivo de vídeo en formato *avi* desexado e preme o botón de subida.
- O controlador da páxina web recibe o arquivo e envíallo xunto co usuario a *VideoService*.
- En *VideoService*, creamos unha entidade de Video co nome do arquivo, o usuario que o subiu e o *flag* de procesado a falso. Dita entidade envíase a *VideoRepository*.
- Neste repositorio, realízase o gardado en base de datos da entidade.

- Gracias ao *ORM*, en *VideoService* xa dispoñemos da entidade que gardamos co id de base de datos. Dito id utilizarase no nome do arquivo de vídeo.
- Realízase a copia do arquivo de vídeo ao directorio *./kaldi/videos/not_processed*.
- Completase o proceso e volve a execución ao controlador da web. Se non se produciu ningún erro, realízase unha redirección á páxina de vídeos subidos.
- En dita páxina, mostrarase unha mensaxe similar a "Número de vídeos subidos: XX. Vídeos á espera de procesar: YY".

2 - Procesado dos vídeos

O seguinte paso a realizar é a obtención da transcrición do vídeo, o que se decidiu que fose controlado por un administrador dado o gran consumo de recursos que produce. A secuencia detallada a continuación pode consultarse na figura 6.6 (páxina 59):

- Un administrador accede á páxina de panel de administración e pulsa o botón de procesado de vídeos.
- O controlador da páxina realiza a chamada a *KaldiService*.
- Na capa de negocio, crease e iniciase un novo proceso que apunta ao arquivo *sh* iniciador do procesado.
- En dito executable, con *FFmpeg* realízase a extracción do audio dos vídeos localizados en *./kaldi/videos/not_processed*. Os arquivos de audio *wav* creados son movidos a *./kaldi/audios*.
- Tras ese procesado, no arquivo *sh*, realízase o traslado dos arquivos de vídeo de *./kaldi/videos/not_processed* a *./kaldi/videos*, para evitar que se mesturen arquivos que foron procesados con outros que non.
- A continuación é a quenda do recoñecedor de voz. Este obtén os audios, realiza as preparacións e procesados necesarios e finalmente garda os resultados da transcrición.
- Tras o procesado do recoñecedor, a execución volve ao arquivo *sh* e realízase o borrado dos audios utilizados.
- Por último, realízase con *FFmpeg* a conversión dos vídeos xa procesados de formato *avi* a *mp4*, e son trasladados á carpeta de recursos da capa web para o seu posterior acceso. Posteriormente, os vídeos en formato *avi* tamén son eliminados.

3 - Indexación das transcricións

O seguinte paso é a indexación das transcricións e a incorporación dos vídeos ao sistema para que poidan ser participes das procuras. A secuencia detallada a continuación pode consultarse na figura 6.7 (páxina 60):

- Un administrador accede á páxina de panel de administración e pulsa o botón de engadido de vídeos.
- O controlador da páxina realiza a chamada a *LuceneService* para iniciar o indexado dos datos.
- Neste servicio, o primeiro que se fai é realizar unha chamada a *KaldiService*, pedindo o arquivo da transcrición.
- En *KaldiService*, obtense o arquivo, corríxese a codificación (de *ISO-8859-1* a *UTF-8*]) e devolvese como un *String*.
- De volta en *LuceneService*, o arquivo divídese en partes, utilizando como divisor os espazos en branco. Para cada parte, compróbase se é un identificador da transcrición (o identificador consta de tres partes, como se pode observar na figura 4.3 (páxina 31)).
- No caso de que non se trate dun identificador, quere dicir que dito *string* se trata dunha das palabras da frase transcrita, polo que se van agregando para posteriormente formar o valor do *field* de Lucene.
- No caso de que se trate dun identificador:

  O novo identificador é tratado para obter o id do vídeo correspondente e gárdase nunha variable.

  No caso de que existise outro id de vídeo anterior no bucle, compróbase se existe en base de datos mediante unha chamada a *VideoRepository*.

  Se dito vídeo existe, márcase como procesado (para permitir ser buscado) e posteriormente gardamos o *field* cos valores: [titulo: id_video, valor: conxunto_palabras].

Cabe destacar que durante este proceso, aparte do procesado dos datos, configúranse outras opcións propias do indexador, que podemos consultar na sección 4.5 relativa a Lucene (páxina 29).

4 - Procura

Para a procura dos vídeos séguese o proceso explicado a continuación:

- O usuario realiza unha busca na web introducindo as palabras desexadas.
- Realízase o camiño ata base de datos para recuperar unha lista dos ids dos vídeos procesados.
- Realízase a busca en *LuceneService* das palabras introducidas polo usuario, comprobando se existen nos *fields* que teñen como título o ids dos vídeos.
- Para os resultados atopados, gárdanse os ids dos vídeos, a partir dos cales se buscarán as entidades correspondentes en base de datos. O número de vídeos recuperados dependerá da paxinación escollida. En este caso, 4 por páxina.

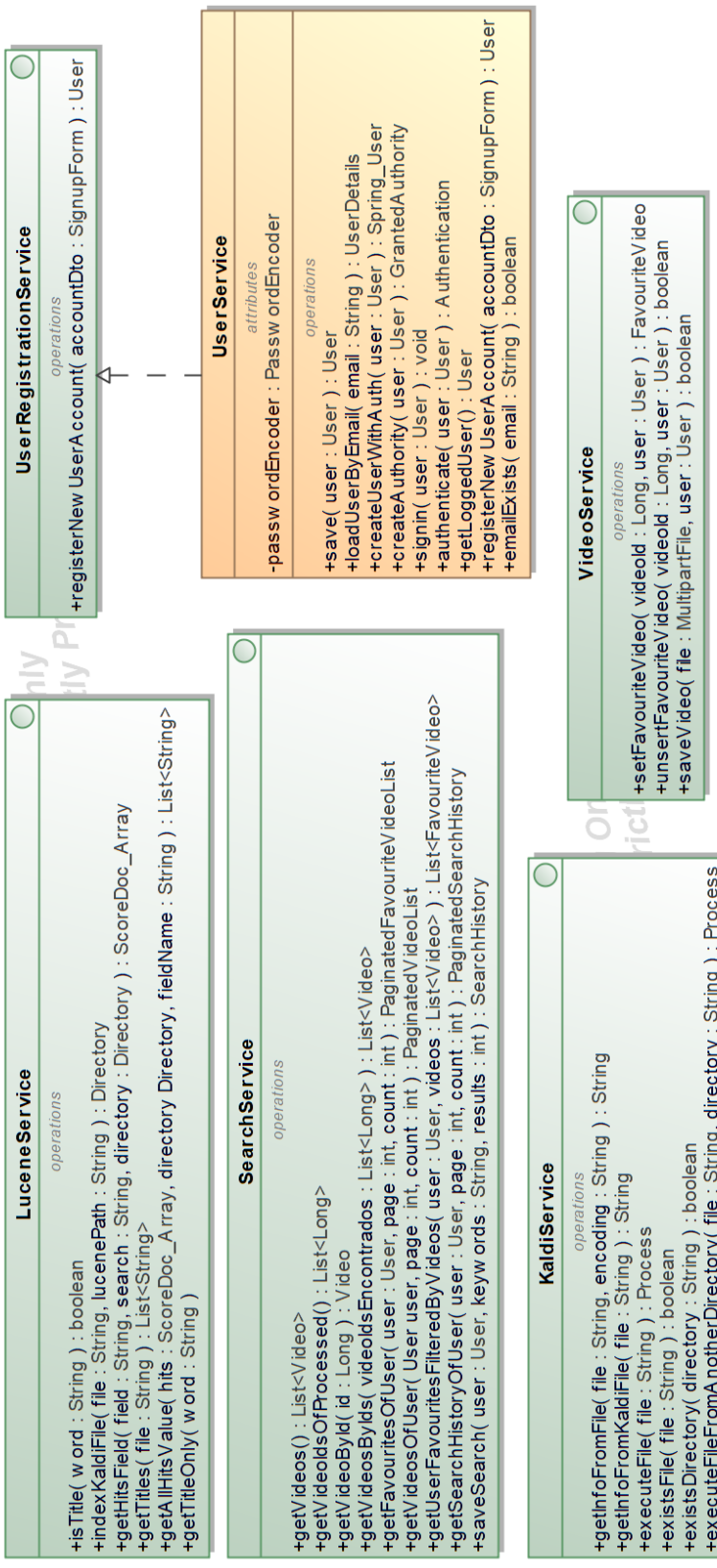


Figura 6.4: Diagrama dos servicios da aplicación.

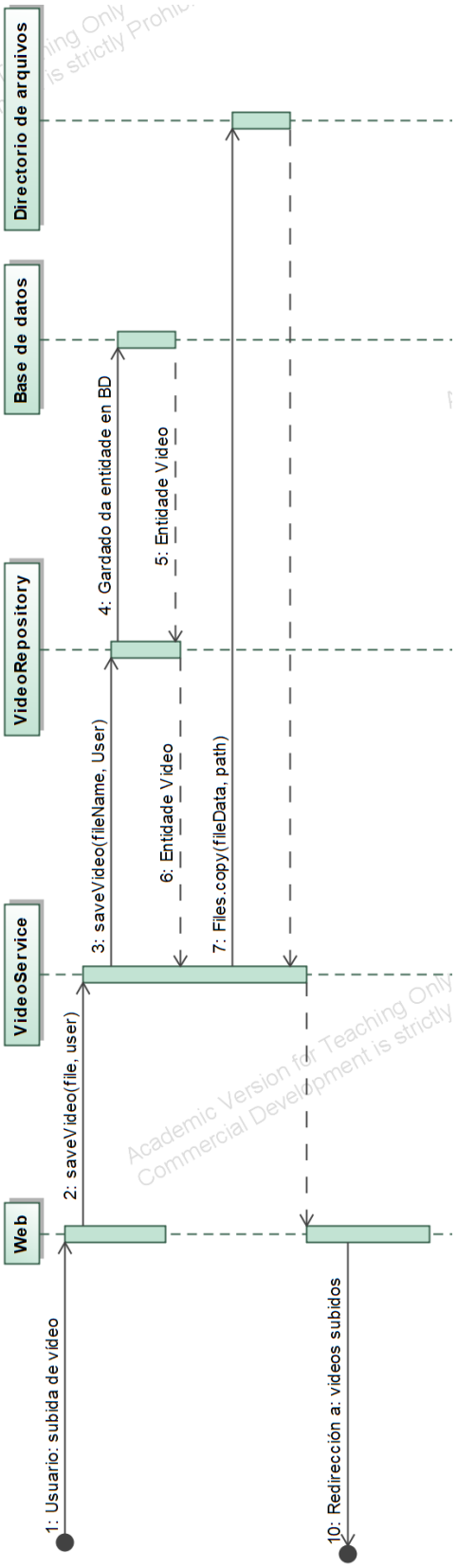


Figura 6.5: Diagrama de secuencia da subida do vídeo.

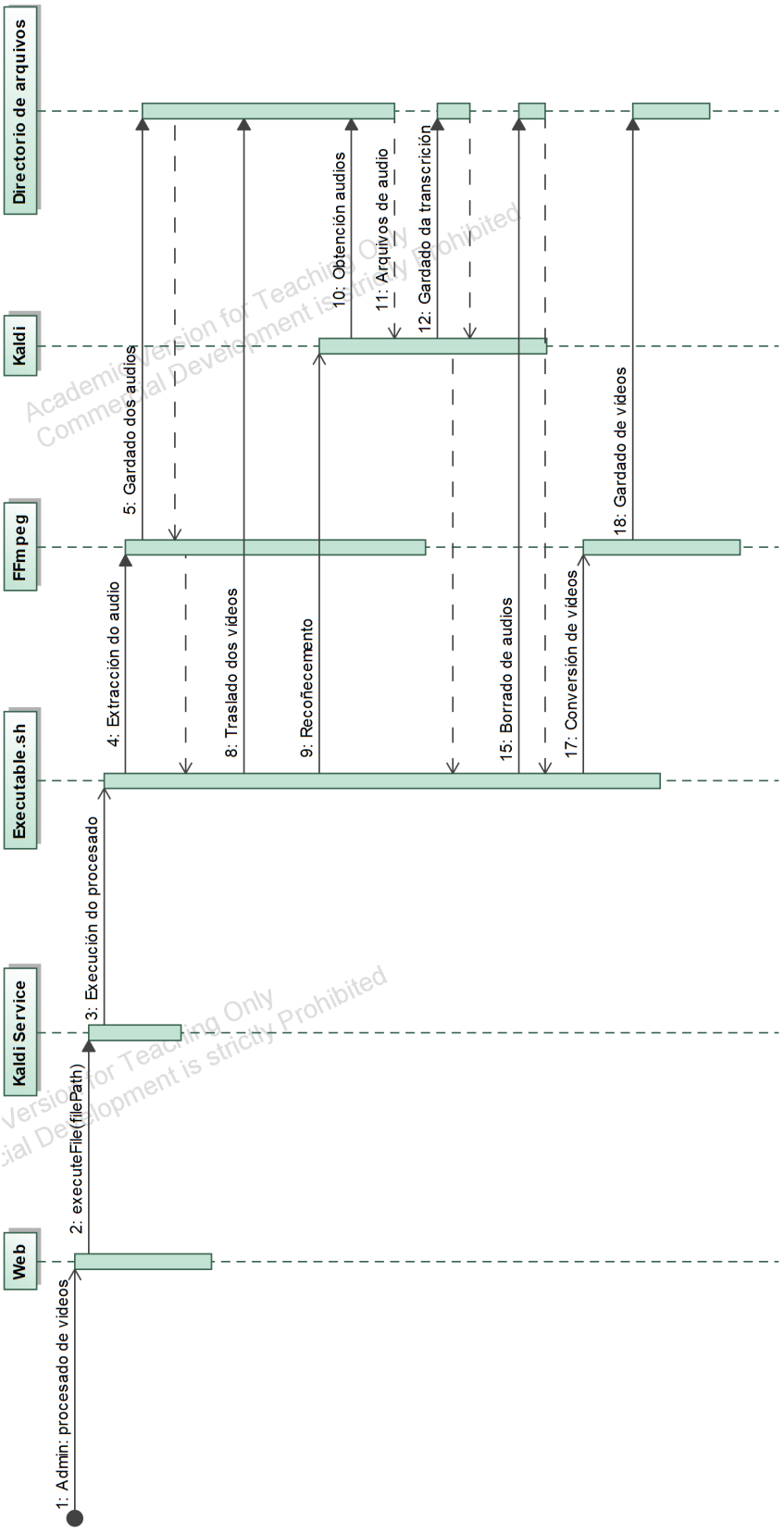


Figura 6.6: Diagrama de secuencia do procesado dos vídeos por parte do recoñecedor de voz.

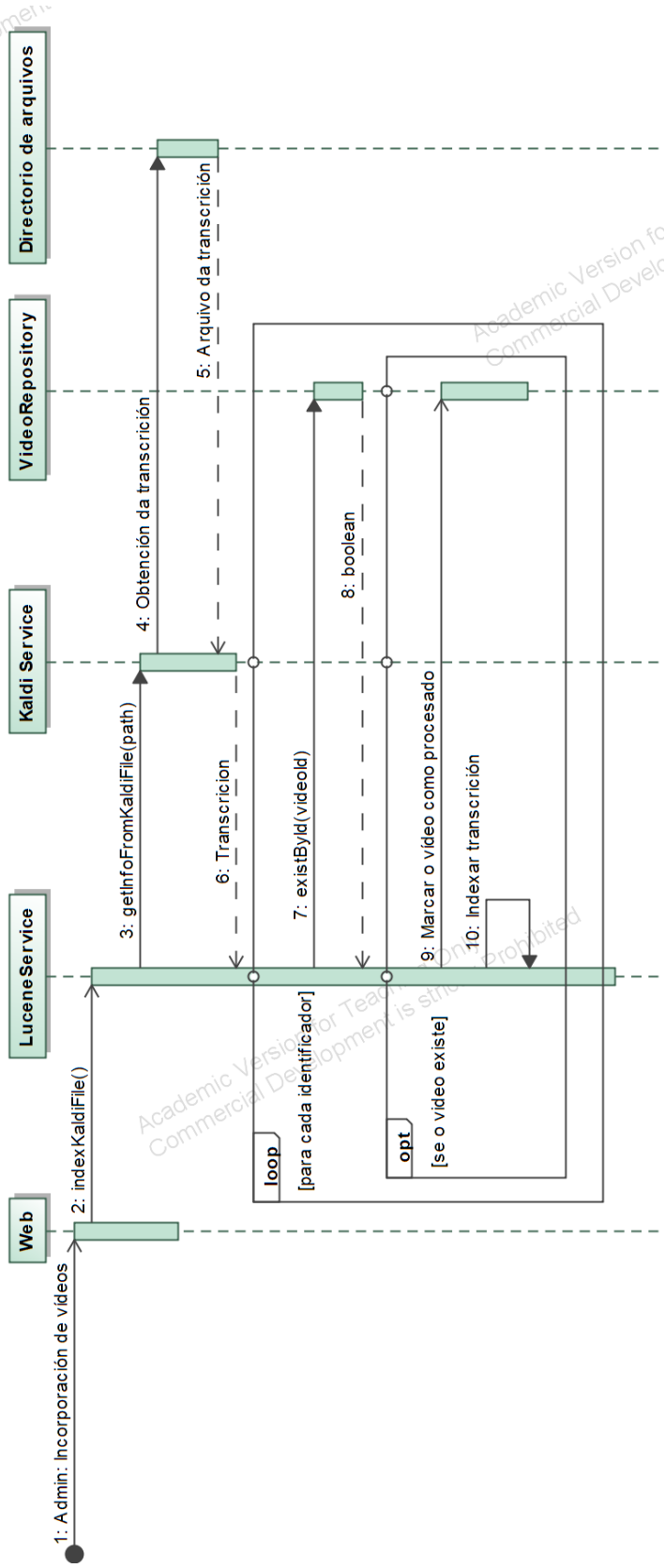


Figura 6.7: Diagrama de secuencia da indexación da transcrición. Tratamento en *LuceneService* parcialmente simplificado

Capítulo 7

Implementación

Neste capítulo veremos a utilización dalgunhas tecnoloxías clave na implementación do proxecto.

7.1 Persistencia

Como se indicou na subsección 4.2.4 do capítulo 4, decidiuse a utilización de bases de datos relacionais xunto cun *ORM*, para o cal foi elixido *Hibernate*, o cal cumpre a especificación de *JPA*. Isto permitiunos definir as nosas entidades directamente en *Java*, as cales foron posteriormente mapeadas automaticamente a base de datos. Para conseguilo, utilizamos as seguintes anotacións:

- **@Entity**: permite definir a clase como unha entidade, a cal será mapeada en base de datos.
- **@Table**: xunto co atributo *name*, definimos o nome da táboa que representará a entidade en base de datos.
- **@Id**: indica que atributo será o que actúe de identificador.
- **@GeneratedValue** e **@SequenceGenerator**: indican o método de xeración de ids utilizado, xunto co nome do xerador.
- **@Column**: permite establecer restricións ás columnas sobre as que se mapean os atributos. Neste caso utilízanse os parámetros *unique* para evitar valores repetidos e *nullable*, para permitir valores nulos.
- **@OneToMany** e **@ManyToOne**: indican as relacións de cardinalidade entre entidades. Co atributo *fetch*, podemos indicar se queremos que as entidades relacionadas se carguen de maneira perezosa (*LAZY*) ou impaciente (*EAGER*).

- **@JoinColumn**: indica con que columna de base de datos se relaciona.

```java
@Entity
@Table(name = "UserProfile")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "UserIdGenerator")
    @SequenceGenerator(name = "UserIdGenerator", sequenceName = "UserSeq")
    @Column(updatable = false, nullable = false)
    private Long id;

    @Column(unique = true)
    private String email;
    private String password;
    private String name;
    private String firstSurname;
    private String secondSurname;
    private String role = "ROLE_USER";

    @Column(unique = true, nullable = true)
    private String dni;
    private String phone;

    @OneToMany(mappedBy = "user", fetch = FetchType.LAZY)
    private List<Video> uploadedVideos = new LinkedList<>();

    @OneToMany(mappedBy = "user", fetch = FetchType.LAZY)
    private List<FavouriteVideo> favouriteVideos = new LinkedList<>();

    @OneToMany(mappedBy = "user", fetch = FetchType.LAZY)
    private List<SearchHistory> searchHistories;
```

Figura 7.1: Exemplo da utilización das anotacións necesarias para a persistencia na entidade *User*.

Por outro lado, para acceder aos obxectos mapeados en base de datos, utilizouse unha implementación de *JpaRepositoy*, *Spring Data Jpa*. Isto proporciónanos funcións básicas *CRUD* para a xestión das entidades, ademais de permitirnos definir outras de maneira sinxela. Se observamos a figura 7.2, podemos destacar varias cousas:

- A anotación *@Repository* indica que esa interface é un *DAO* ou obxecto de acceso a datos.
- A anotación *@Query* permite escribir unha consulta a base de datos utilizando os parámetros da función.

  No caso de que necesitemos paxinación, é necesario definir a consulta co parámetro *value*, mentres que definimos outra similar co parámetro *countQuery* para recuperar o número de resultados da consulta.
- Para funcións sinxelas, é posible non definir ningunha consulta, xa que *JPA* interpreta

as nosas necesidades en base ao atributo que se lle pasa na función e ao tipo de valor de saída, como se pode observar na función `int countByUser (User user)`.

```
@Repository
public interface VideoRepository extends JpaRepository<Video, Long> {

    @Query(value = "select v from Video v where v.user = :user",
            countQuery = "select count(v) from Video v where v.user = :user")
    List<Video> getByUser(User user, Pageable pageable);

    @Query(value = "select v from Video v where v.user = :user AND processed = true",
            countQuery = "select count(v) from Video v where v.user = :user AND processed = true")
    List<Video> getByUserProcessed(User user, Pageable pageable);

    int countByUser(User user);

    @Query("select count(v) from Video v where v.user = :user AND processed = false")
    int countByUserNotProcessed(User user);

    @Query("select v.id from Video v where v.processed = true ORDER BY v.id")
    List<Long> getVideoIdsOfProcessed();

    @Query("select v from Video v where v.id IN :ids ORDER BY v.id")
    List<Video> getVideosByIds(List<Long> ids);

}
```

Figura 7.2: Exemplo dun repositorio que estende de *JpaRepository*.

7.2 Seguridade na capa web

7.2.1 Spring Security

Para controlar o acceso ás distintas seccións da web, utilizouse *SpringSecurity*, ao que se lle sobrescribiu o comportamento por defecto para adaptalo ás necesidades do proxecto. A política que se utilizou foi a seguinte:

- Permitir o acceso ao panel de administración só aos administradores.
- Aos usuarios non rexistrados, por defecto o acceso, para posteriormente permitirlle certas páxinas: *login*, rexistro, páxina de inicio, resultados de busca, e vídeo, así como aos recursos necesarios para a correcta visualización da aplicación.
- Aos usuarios rexistrados, permíteselle o acceso a todo o que non estea expresamente prohibido.

7.2.2 SSL e HTTPS

Coa dobre función de aumentar a seguridade e permitir o correcto funcionamento da *Web Speech Api*, engadiuse un certificado *SSL* auto asinado, xunto coa redirección das pe-

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests().antMatchers("/admin/**").access("hasRole('ROLE_ADMIN')")
            .antMatchers("/", "/login", "/signup", "/home", "/video", "/resultados", "/static/**", "/css/**",
                    "/js/**", "/video/**", "/registration")
            .permitAll().anyRequest().authenticated().and().formLogin().loginPage("/login").permitAll()
            .failureUrl("/login?error=1").loginProcessingUrl("/authenticate").and().logout().logoutUrl("/logout")
            .permitAll().logoutSuccessUrl("/");
    http.csrf().disable();
}
```

Figura 7.3: Configuración do acceso ás *URLs* da aplicación.

ticións *HTTP* a *HTTPS*. Para iso, creouse unha *KeyStore* en formato *PKCS12* co comando: `keytool -genkeypair -alias tomcat -storetype PKCS12 -keyalg RSA -keysize 2048 -keystore keystore.p12 -validity 3650`, ao que se lle indicaron os datos requiridos.

Posteriormente, no arquivo *application.properties*, engadiuse a configuración necesaria para a utilización de dita *KeyStore* e para activar *SSL*.

```
1  server.port=8443
2  server.ssl.key-store = classpath:keystore.p12
3  server.ssl.key-store-password = password
4  server.ssl.key-store-type = PKCS12
5  server.ssl.key-alias = tomcat
```

Listing 7.1: Configuración aplicada en application.properties para a securización

En este punto, xa funcionaba o servidor mediante *HTTPS* no porto 8443, pero se intentábamos entrar por *HTTP*, non era posible. Como solución a ese problema, implementouse a redirección dende o porto *HTTP* (8080) a *HTTPS* (8443). Para iso, é necesario engadir no arquivo *application.properties*, o cal que modificamos antes, a liña `http.port=8080`. Posteriormente, definimos a regra para a redirección da conexión e engadímoslla ao *Servlet* de *Tomcat*, como podemos observar na figura 7.4.

7.3 Interface de usuario

Na implementación da interface, buscouse un deseño simple e pouco recargado. A utilización de *Materialize* xunto con *Thymelaf* (subsección 4.4.4), permitiu realizar unhas páxinas sinxelas e atractivas visualmente, como podemos apreciar nas seguintes capturas da aplicación.

```java
@SpringBootApplication
public class VideosearchApplication extends SpringBootServletInitializer {

    public static void main(String[] args) {
        SpringApplication.run(VideosearchApplication.class, args);
    }

    @Bean
    public ServletWebServerFactory servletContainer() {
        TomcatServletWebServerFactory tomcat = new TomcatServletWebServerFactory() {
            @Override
            protected void postProcessContext(Context context) {
                SecurityConstraint securityConstraint = new SecurityConstraint();
                securityConstraint.setUserConstraint("CONFIDENTIAL");
                SecurityCollection collection = new SecurityCollection();
                collection.addPattern("/*");
                securityConstraint.addCollection(collection);
                context.addConstraint(securityConstraint);
            }
        };
        tomcat.addAdditionalTomcatConnectors(redirectConnector());
        return tomcat;
    }

    private Connector redirectConnector() {
        Connector connector = new Connector("org.apache.coyote.http11.Http11NioProtocol");
        connector.setScheme("http");
        connector.setPort(8080);
        connector.setSecure(false);
        connector.setRedirectPort(8443);
        return connector;
    }
}
```

Figura 7.4: Configuración da redirección de *HTTP* a *HTTPS*.

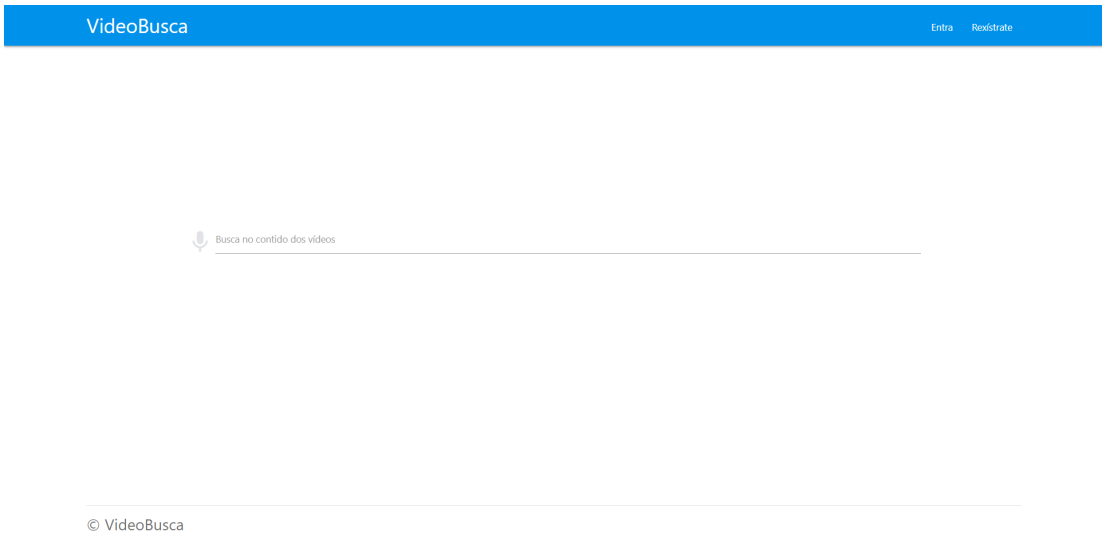


Figura 7.5: Páxina de inicio da aplicación para un usuario non rexistrado.

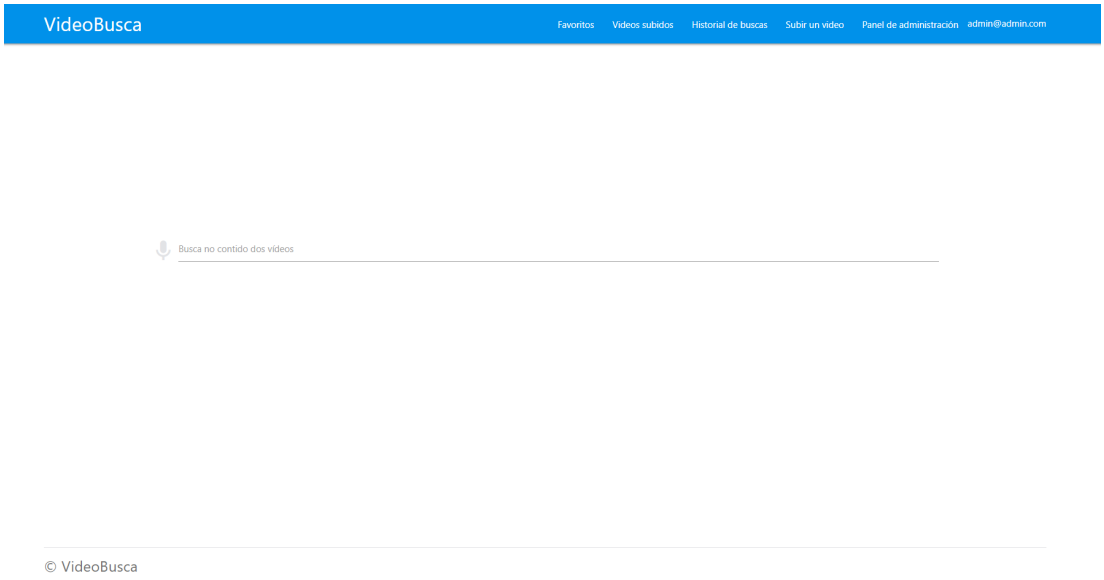


Figura 7.6: Páxina de inicio da aplicación para un administrador.

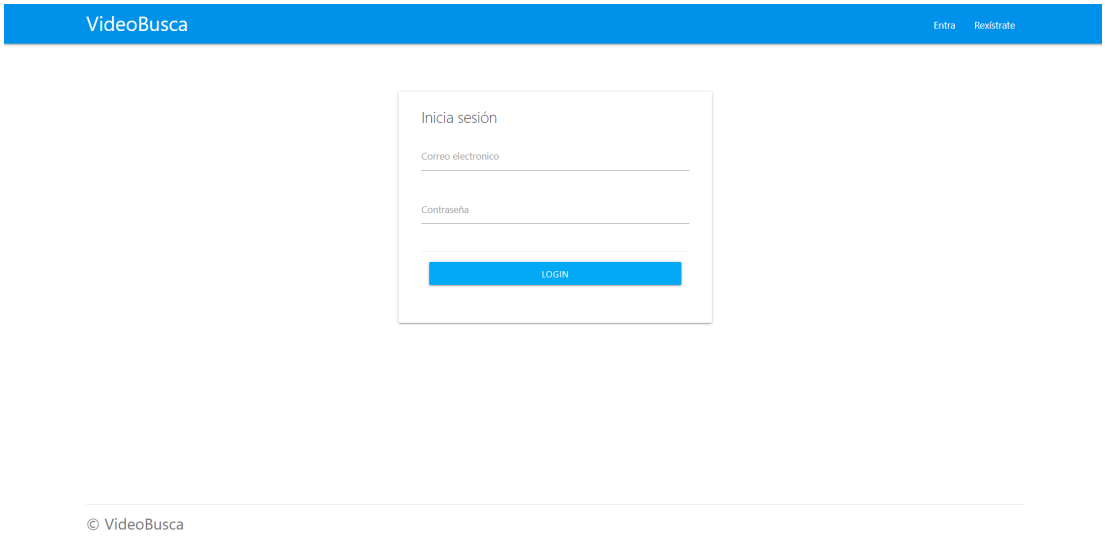


Figura 7.7: Páxina de *login*.

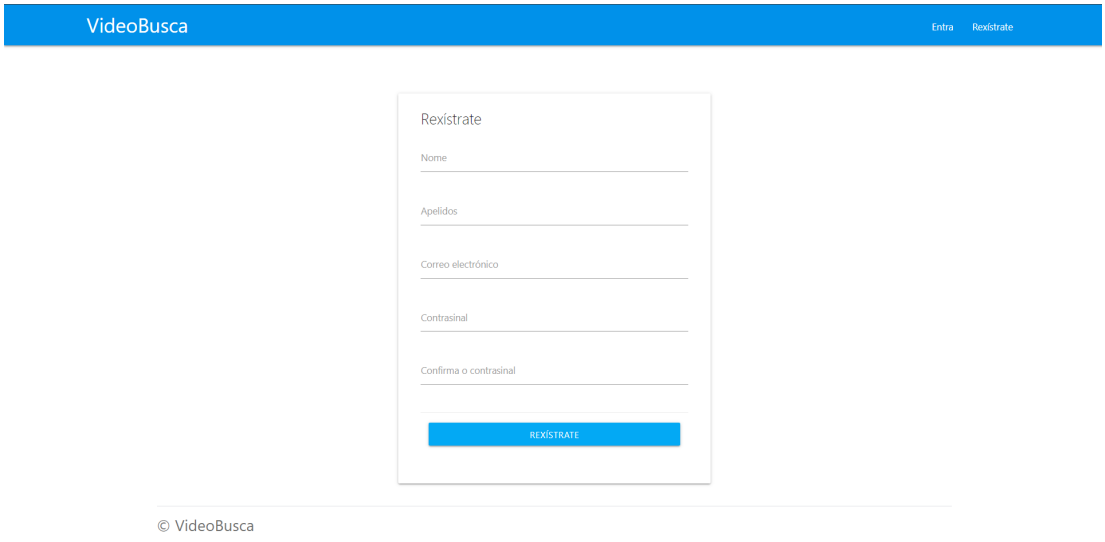


Figura 7.8: Páxina de rexistro.

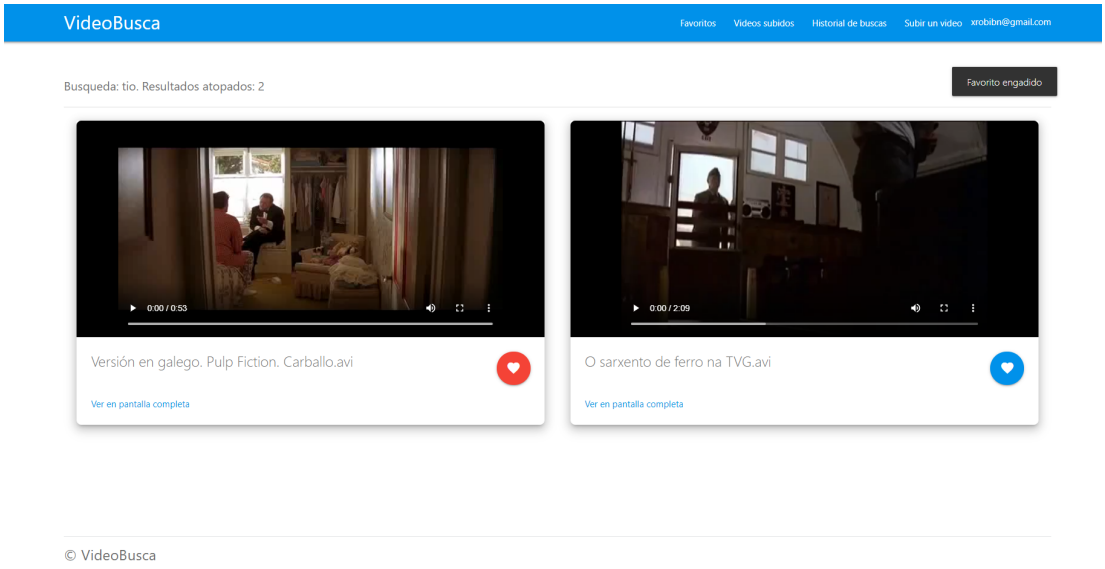


Figura 7.9: Páxina de resultados de busca con dous vídeos. Aparece unha pequena notificación froito de marcar un vídeo como favorito.

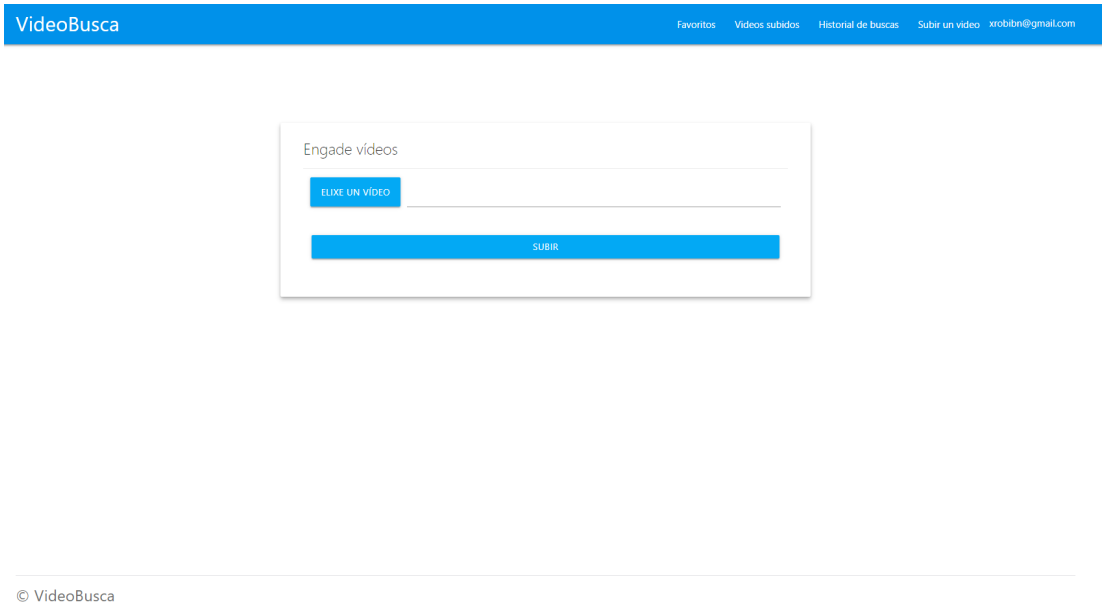


Figura 7.10: Páxina para a subida de vídeos.

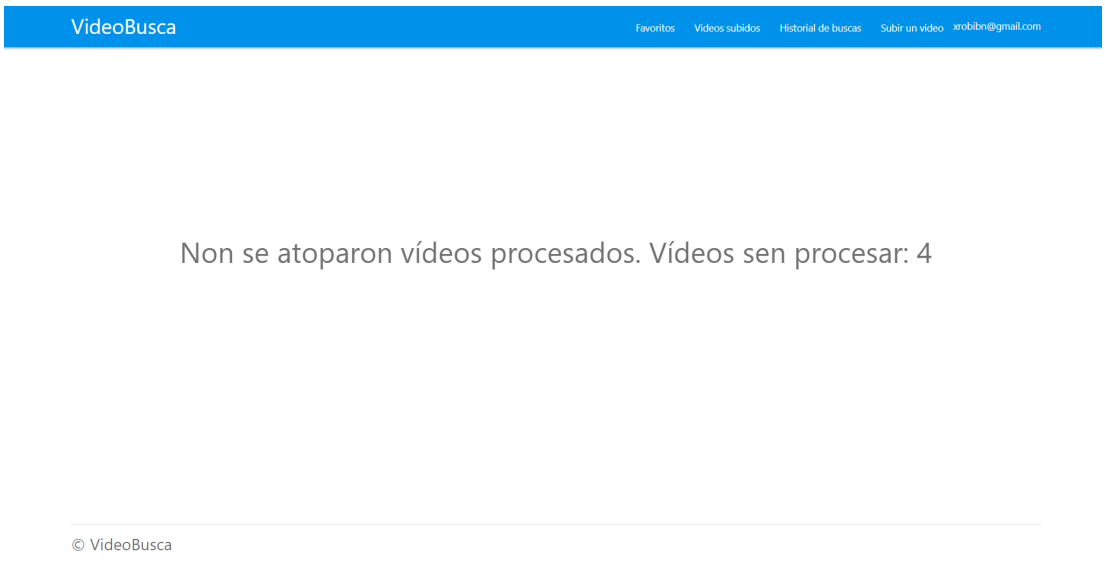


Figura 7.11: Páxina de vídeos subidos pero sen ningún procesado polo momento.

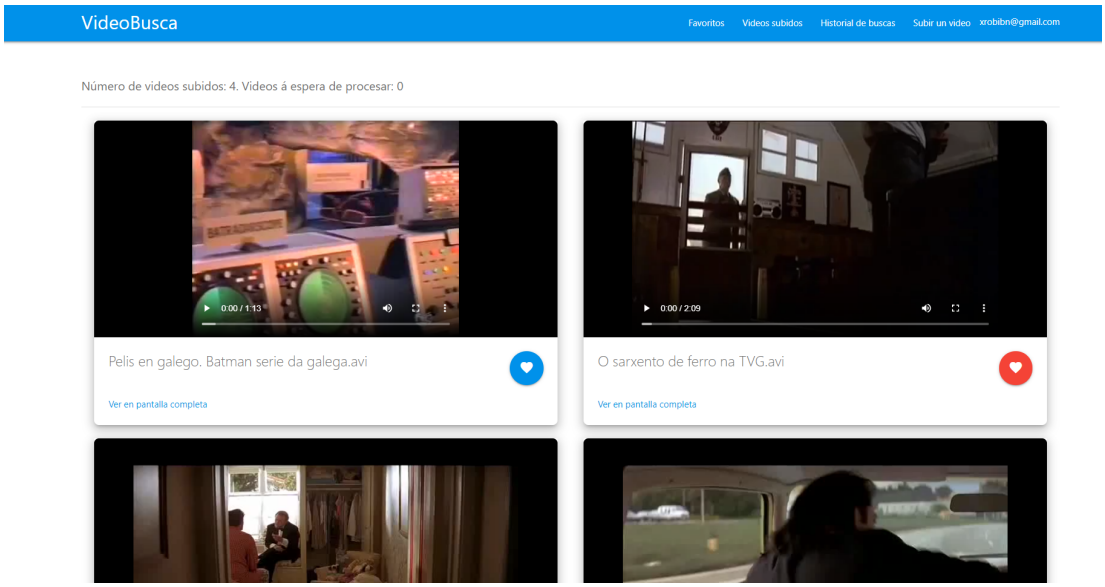


Figura 7.12: Páxina de vídeos subidos con catro resultados, un deles marcado como favorito.

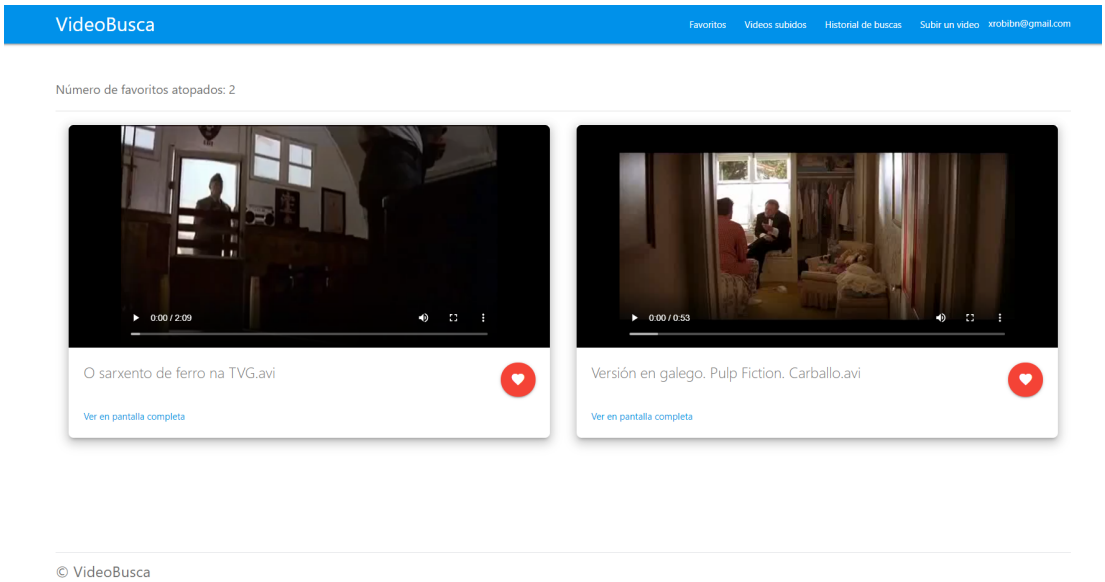


Figura 7.13: Páxina de favoritos con dous vídeos.

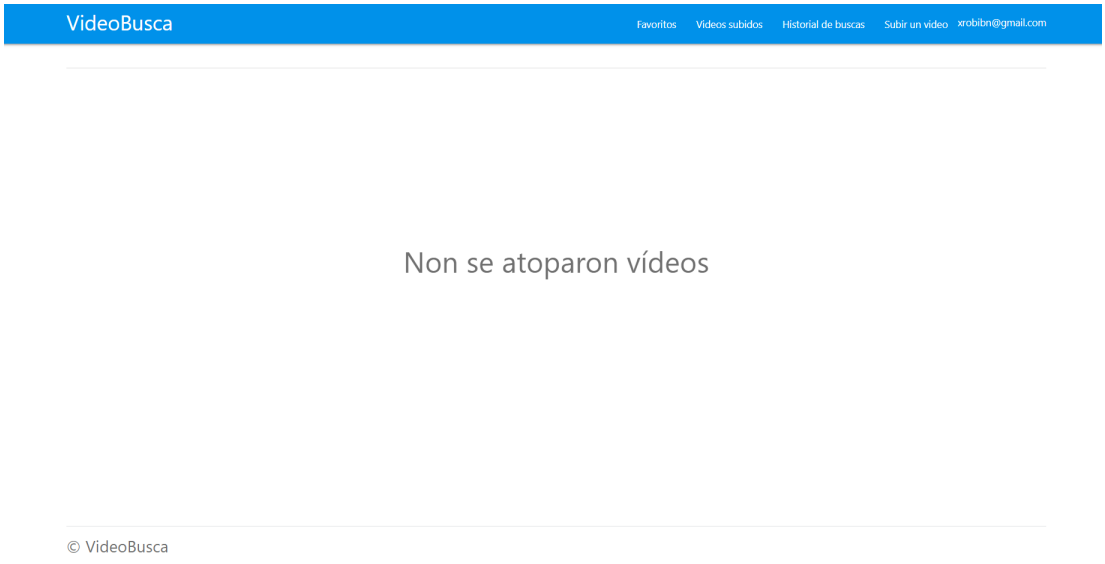


Figura 7.14: Páxina de favoritos sen ningún vídeo.

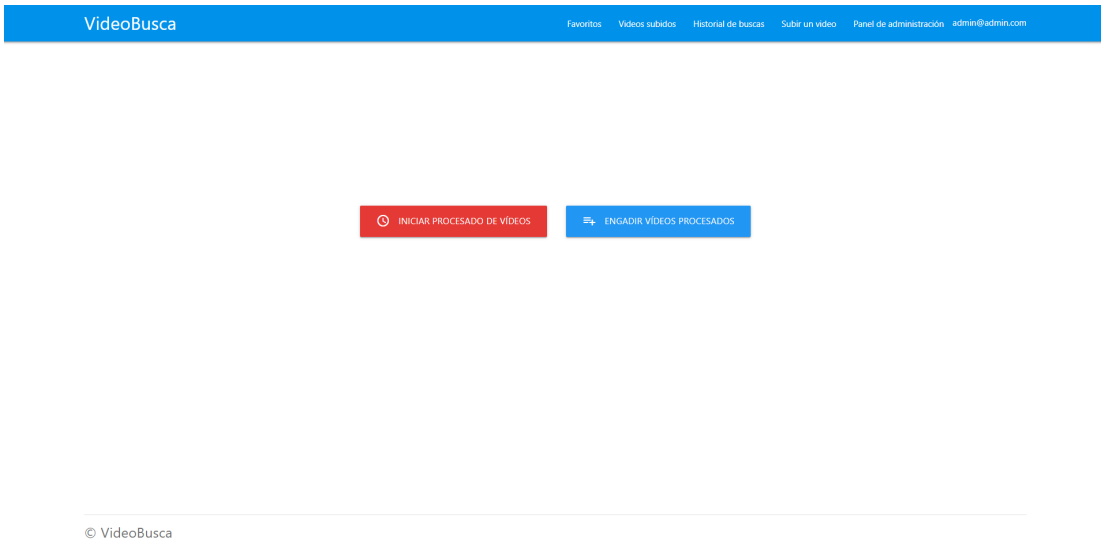


Figura 7.15: Panel de administración.

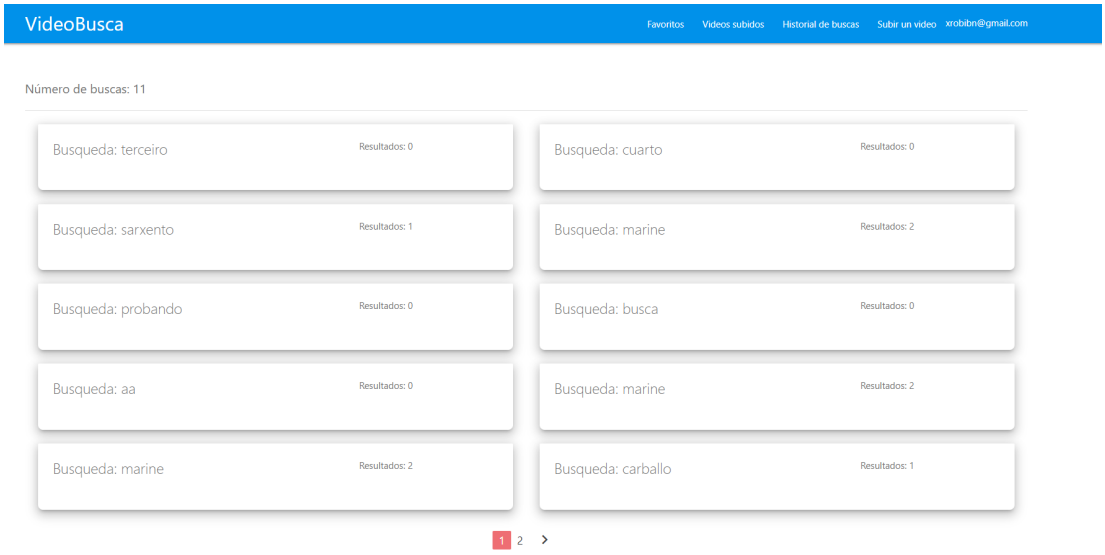


Figura 7.16: Páxina do historial de buscas con paxinación.

Capítulo 8
Conclusións

Neste capítulo final da memoria, presentarase a situación final do proxecto, as liñas futuras e as leccións aprendidas.

8.0.1 Situación do proxecto e liñas futuras

O estado final do proxecto é existoso, aínda que non perfecto. Os requisitos descritos na sección 6.1 foron conseguidos, sen embargo a xestión e organización non foi tan boa como debería, sobre todo en dous puntos:

- Debido a circunstancias laborais e persoais, non se conseguiu unha constancia entre *sprints*, como se explicou na adaptación ao proxecto de *Scrum* 5.2.
- Pese a que a inspección continua con *SonarQube* estaba correctamente configurada e se realizaron os cambios propostos en diversas ocasións, non se foi todo o constante e exhaustivo que se debería.

En canto a liñas futuras a seguir, as máis claras serían as seguintes:

- Permitir o procesado de vídeos de diversas extensións, xa que polo de agora só se poden subir arquivos *avi*.
- Automatizar o procesado de vídeos. Isto actualmente non é posible por temas de rendemento. No caso de contar cun servidor suficientemente potente, deberíase valorar.
- Mellor xestión de usuarios e vídeos (CRUD).
- Internacionalización da aplicación, utilizando á súa vez os recoñecedores de voz necesarios para cada idioma.
- Comezar a reproducir o vídeo no instante no que se pronunciou a frase atopada.

- Integrar solucións relativas a *Spoken Document Retrieval (SDR)* e cambiar a API que se utiliza para a busca por voz dende web, por un sistema eficiente de *Query-by-Example Spoken Document Retrieval (QbESDR)*.

8.0.2 Leccións aprendidas

Con este proxecto, experimentouse por primeira vez o desenvolvemento dunha aplicación de principio a fin, consolidando os coñecementos de xestión, análise, deseño e programación, así como o uso de *Scrum* e das tecnoloxías xa coñecidas. Por outro lado, tamén se realizou unha primeira inclusión no complexo mundo das tecnoloxías de recoñecemento da fala, xunto coas de recuperación da información. Finalmente, asumir que durante todo o transcurso do proxecto, foron atopados erros, pero tamén foron atopadas solucións, producindo unha evolución profesional.

Apéndices

Relación de Acrónimos

ACID Atomicidad, consistencia, aislamiento, durabilidad.

AJAX Asynchronous JavaScript And XML.

AOP Aspect-Oriented Programming.

ASR Automatic Speech Recognition.

BD Base de datos.

BSD Berkeley Software Distribution.

CD Continuous Delivery.

CI Continuous Integration.

CPU Central Processing Unit.

CRUD Create, Read, Update, Delete.

CSS Cascading Style Sheets.

DAO Data Access Object.

DoD Definition of Done.

DNN Deep Neural Network.

FSTs Finite State Transducers.

GPU Graphics Processing Unit.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IoC Inversion of Control.

JPA Java Persistence Api.

LVCSR Large Vocabulary Continuous Speech Recognition.

MVC Modelo-Vista-Controlador.

MVCC Multi-Version Concurrency Control.

ORM Object-relational Mapping.

OOV Out-of-vocabulary.

PiP Picture in Picture.

POJO Plain Old Java Object.

POM Project Object Model.

QbESDR Query-by-Example Spoken Document Retrieavl.

RBMs Restricted Boltzmann Machines.

SDR Spoken Document Retrieval.

SSD Solid State Disk.

SSL Secure Sockets Layer.

TDD Test-driven development.

URL Uniform Resource Locator.

Glosario

Bean Compoñente de *Java* utilizado para encapsular varios obxectos nun só.

Indice invertido Os índices invertidos son estruturas de datos que se utilizan para indexar e recuperar información de maneira eficiente. En vez de ter un índice de documentos e posteriormente buscar o dato, temos un índice cos datos, que vai asociado a unha colección de documentos nos que se encontra dito dato.

HMM (modelo oculto de Markov) modelo estatístico no que se asume que o sistema a modelar é un proceso de Markov de parámetros descoñecidos.

JDBC *Java Database Connectivity*, é unha *API* que permite a execución de operacións de bases de datos dende *Java* mediante un dialecto *SQL*.

MFCC (coeficientes cepstrales das frecuencias de Mel) Trátase de coeficientes utilizados para a representación da fala baseados na percepción humana dos sons.

Stemming É o proceso de reducir unha palabra á súa raíz ou *stem*. Por exemplo, se temos as palabras *pedrolo* e *pedra*, ambas serán reducidas á mesma raíz: *pedr*.

Stop words (palabras baleiras) Trátanse de palabras de uso común que normalmente carecen de gran significado, como por exemplo os artigos ou os pronomes, polo que normalmente se filtran e ignoran ao procesar texto. Desta maneira mellóranse os resultados da busca e aforrase tempo de procesamento e espazo de almacenamento.

Transformada de Fourier Transformación matemática que permite relacionar unha función no dominio do tempo cunha función no dominio da frecuencia, permitindo a conversión entre ambos dominios.

Bibliografía

[1] C. Gaida, P. Lange, R. Petrick, P. Proba, and D. Suendermann-Oeft, "Comparing open-source speech recognition toolkits ⊠," p. 12. [En liña]. Dispoñible en: http://suendermann.com/su/pdf/oasis2014.pdf

[2] Hey siri: An on-device DNN-powered voice trigger for apple's personal assistant. [En liña]. Dispoñible en: https://machinelearning.apple.com/research/hey-siri

[3] Keyword spotting for google assistant using contextual speech recognition. [En liña]. Dispoñible en: https://storage.googleapis.com/pub-tools-public-publication-data/pdf/be2559f953dce47e69f4d06692df1184719c4d4b.pdf

[4] A. Chhetri, P. Hilmes, T. Kristjansson, W. Chu, M. Mansour, X. Li, and X. Zhang, "Multichannel audio front-end for far-field automatic speech recognition," in *2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE, pp. 1527–1531. [En liña]. Dispoñible en: https://ieeexplore.ieee.org/document/8553149/

[5] Course: ELEC-e5510 - speech recognition l, 28.10.2015-04.12.2015, section: Lectures. [En liña]. Dispoñible en: https://mycourses.aalto.fi/course/view.php?id=5180&section=2

[6] R. a. S. I. S. R. International, "Speech recognition using monophone and triphone based continuous density hidden markov models." [En liña]. Dispoñible en: https://www.academia.edu/19039608/Speech_Recognition_Using_Monophone_and_Triphone_Based_Continuous_Density_Hidden_Markov_Models

[7] joseangl, "Modelos probabilísticos: Aplicación al reconocimiento automático del …." [En liña]. Dispoñible en: https://es.slideshare.net/joseangl/charla-linares

[8] M. YU, "Capstone development," original-date: 2014-10-28T03:04:11Z. [En liña]. Dispoñible en: https://github.com/yufree/Capstone

[9] What is inverted index? it is a well known fact that you need to build indexes to implement efficient searches. what is the difference between index and inverted index,

and how does one build inverted index? - quora. [En liña]. Dispoñible en: https://www.quora.com/What-is-inverted-index-It-is-a-well-known-fact-that-you-need-to-build-indexes-to-implement-efficient-searches-What-is-the-difference-between-index-and-inverted-index-and-how-does-one-build-inverted-index

[10] A. Hauptmann, "Automatic spoken document retrieval," p. 10.

[11] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, "The kaldi speech recognition toolkit," in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, event-place: Hilton Waikoloa Village, Big Island, Hawaii, US. [En liña]. Dispoñible en: http://publications.idiap.ch/downloads/papers/2012/Povey_ASRU2011_2011.pdf

[12] Can i use... support tables for HTML5, CSS3, etc. [En liña]. Dispoñible en: https://caniuse.com/#search=Speech

[13] ¿qué es un java maven artifact ? [En liña]. Dispoñible en: https://www.arquitecturajava.com/que-es-un-java-maven-artifact/

[14] What are the tradeoffs between leveraging kaldi and HTK for speech recognition applications? - quora. [En liña]. Dispoñible en: https://www.quora.com/What-are-the-tradeoffs-between-leveraging-Kaldi-and-HTK-for-speech-recognition-applications

[15] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 4087–4091. [En liña]. Dispoñible en: http://ieeexplore.ieee.org/document/6854370/

[16] J. Li, R. Zhao, Z. Chen, C. Liu, X. Xiao, G. Ye, and Y. Gong, "Developing far-field speaker system via teacher-student learning," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 5699–5703. [En liña]. Dispoñible en: https://ieeexplore.ieee.org/document/8462209/

[17] T. Be'ery, "Open sesame: Picking locks with cortana." [En liña]. Dispoñible en: https://www.slideshare.net/TalBeery1/open-sesame-picking-locks-with-cortana

[18] M. Wu, S. Panchapagesan, M. Sun, J. Gu, R. Thomas, S. N. Prasad Vitaladevuni, B. Hoffmeister, and A. Mandal, "Monophone-based background modeling for two-stage on-device wake word detection," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 5494–5498. [En liña]. Dispoñible en: https://ieeexplore.ieee.org/document/8462227/

[19] A. Raju, S. Panchapagesan, X. Liu, A. Mandal, and N. Strom, "Data augmentation for robust keyword spotting under playback interference." [En liña]. Dispoñible en: http://arxiv.org/abs/1808.00563

[20] H. D. B. Marta Ruiz Costa-jussà, "Reconocimiento automático del habla." [En liña]. Dispoñible en: https://www.exabyteinformatica.com/uoc/Audio/Procesamiento_de_audio/Procesamiento_de_audio_(Modulo_7).pdf

[21] S. Ahmed and G. Buttazzo, "Automatic phoneme recognition using mel-frequency cepstral coefficient and dynamic time warping," p. 68. [En liña]. Dispoñible en: https://core.ac.uk/download/pdf/79623651.pdf

[22] A first take at building an inverted index. [En liña]. Dispoñible en: https://nlp.stanford.edu/IR-book/html/htmledition/a-first-take-at-building-an-inverted-index-1.html

[23] M. Larson and G. J. F. Jones, "Spoken content retrieval: A survey of techniques and technologies," vol. 5, no. 4, pp. 235–422, publisher: Now Publishers, Inc. [En liña]. Dispoñible en: https://www.nowpublishers.com/article/Details/INR-020

[24] P. Lopez-Otero, J. Parapar, and A. Barreiro, "Statistical language models for query-by-example spoken document retrieval," vol. 79, no. 11, pp. 7927–7949. [En liña]. Dispoñible en: https://doi.org/10.1007/s11042-019-08522-z

[25] P. Lopez-Otero, A. Barreiro, and J. Parapar, "Efficient query-by-example spoken document retrieval combining phone multigram representation and dynamic time warping," vol. 56, no. 1, pp. 43–60. [En liña]. Dispoñible en: http://www.sciencedirect.com/science/article/pii/S0306457318301614

[26] P. Lopez-Otero and L. D. Fernández, "GTM-IRLab systems for albayzin 2018 search on speech evaluation," in *IberSPEECH*. [En liña]. Dispoñible en: https://pdfs.semanticscholar.org/c630/20598da8bdb59b540e05b3f527d1d05f65a6.pdf

[27] K. Vesely, A. Ghoshal, L. Burget, and D. Povey, "Sequence-discriminative training of deep neural networks," p. 5. [En liña]. Dispoñible en: https://www.danielpovey.com/files/2013_interspeech_dnn.pdf

[28] D. Can and M. Saraclar, "Lattice indexing for spoken term detection," vol. 19, no. 8, pp. 2338–2347. [En liña]. Dispoñible en: http://ieeexplore.ieee.org/document/5752829/

[29] G. Chen, O. Yilmaz, J. Trmal, D. Povey, and S. Khudanpur, "Using proxies for OOV keywords in the keyword search task," in *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*. IEEE, pp. 416–421. [En liña]. Dispoñible en: http://ieeexplore.ieee.org/document/6707766/

[30] K. Wedekind. HTML5 speech recognition API. [En liña]. Dispoñible en: https://codeburst.io/html5-speech-recognition-api-670846a50e92

[31] O. Taguacundo, "Software libre y software comercial." [En liña]. Dispoñible en: https://www.slideshare.net/taytoemanuel1/software-libre-y-software-comercial-62587597

[32] PostgreSQL: ¿qué es? características, ventajas y desventajas. [En liña]. Dispoñible en: https://hostingpedia.net/postgresql.html

[33] Hibernate with PostgreSQL - 6 things you need to know. [En liña]. Dispoñible en: https://thoughts-on-java.org/hibernate-postgresql-5-things-need-know/

[34] HTML5 video preload | high performance web sites. [En liña]. Dispoñible en: https://www.stevesouders.com/blog/2013/04/12/html5-video-preload/

[35] Apache lucene - apache lucene core. [En liña]. Dispoñible en: https://lucene.apache.org/core/

[36] Apache lucene. [En liña]. Dispoñible en: https://www.ionos.es/digitalguide/servidores/configuracion/apache-lucene/

[37] 262588213843476. using ffmpeg to extract audio from video files. [En liña]. Dispoñible en: https://gist.github.com/protrolium/e0dbd4bb0f1a396fcb55

[38] Atlassian. Why git | atlassian git tutorial. [En liña]. Dispoñible en: https://www.atlassian.com/git/tutorials/why-git

[39] Guia salarial sector TI galicia 2015-2016 - [PDF document]. [En liña]. Dispoñible en: https://vdocuments.site/guia-salarial-sector-ti-galicia-2015-2016.html