



Universidad de Castilla-La Mancha

Escuela Superior de Ingeniería Informática

## **Trabajo Fin de Máster**

Máster Universitario en Ingeniería Informática

# **SERENDIPITY: Servicio web para la recomendación de playlists a partir de una playlist**

*Miguel Ángel Cantero Vllora*

Septiembre, 2021





## TRABAJO FIN DE MÁSTER

Máster Universitario en Ingeniería Informática

# SERENDIPITY: Servicio web para la recomendación de playlists a partir de una playlist

*English title: SERENDIPITY, Web service for recommending  
playlists from a playlist*

**Autor:** Miguel Ángel Cantero Villora

**Tutores:** José Antonio Gámez Martín y Juan Ángel Aledo Sánchez

Septiembre, 2021



*A mis padres*



## Declaración de autoría

Yo, Miguel Ángel Cantero Vállora con DNI 47072961-B, declaro que soy el único autor del Trabajo Fin de Máster titulado "SERENDIPITY: Servicio web para la recomendación de playlists a partir de una playlist " y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Albacete, a 10 de septiembre de 2021

Fdo.: Miguel Ángel Cantero Vállora





## Resumen

En la actualidad, compartir y descubrir nueva música resulta muy sencillo dada la cantidad de servicios de música en streaming (*Spotify*, *YouTube Music*, *Apple Music*...) o redes sociales musicales (*Last.fm*).

Antes del auge de Internet y la aparición de los servicios musicales, las formas más comunes de descubrir música se limitaban a la radio y al intercambio de medios (como casetes o CDs). Centrándonos en ésta última forma, era muy común intercambiar recopilaciones entre amigos con gustos similares para descubrir nuevas canciones. Estas grabaciones, en algunos casos, podían ser recopilaciones especiales para reproducir mientras se realizaban actividades como pueden ser hacer deporte, estudiar, viajar, etc. Actualmente este tipo de recopilaciones han sido sustituidas por playlists musicales, donde cada usuario añade las canciones que le gustan bajo un título que las identifique.

La gran mayoría de los servicios musicales hacen muy sencillo buscar una canción similar a la que se está reproduciendo, y también ofrecen recomendaciones de emisoras (bloques de canciones, que pueden ser aleatorias, relacionadas con el género de la canción o con artistas relacionados). Sin embargo, es muy poco frecuente ofrecer playlists de otros usuarios como recomendación a otra playlist que estamos escuchando.

El objetivo de este Trabajo de Fin de Máster es proporcionar un sistema que, dada una playlist, mediante su título, canciones que la conforman y otras características dadas, encuentre otras playlists similares entre los distintos usuarios de *Spotify* para ofrecer al usuario.



## Abstract

Nowadays, sharing and discovering new music is very easy, given the number of music streaming services such as *Spotify*, *YouTube Music*, *Apple Music*, ... or social music networks like *Last.fm*.

Before the rise of the Internet and the emergence of music streaming services, the most common ways to discover new music were limited to radio and media sharing (such as cassettes or CDs). Focusing on this last way, it was very common to exchange compilations between friends with similar tastes to discover new songs. These recordings, in some cases, could be special compilations to reproduce while carrying out activities such as doing sports, studying, traveling, ... Currently this type of compilation has been replaced by musical playlists, where each user adds the songs they like grouped by a given title related with the content.

All of the music streaming services provide us with very easy ways for finding related song to the one that is currently playing, and they also offer recommendations with the radios feature (blocks of songs, which can be randomly selected, related to the genre of the song or related artists). However, it is not so common to offer playlists from other users as a recommendation to a given playlist.

The main aim of this Master of Science thesis is to provide a system that, given a playlist, through its title, their songs and other characteristics, finds other similar playlists among the different *Spotify* users to offer to a given user.



## Agradecimientos

Ante todo, me gustaría agradecer a mis tutores, José Antonio y Juan Ángel, toda la ayuda que me han ofrecido durante la realización de este TFM, ya que sin ellos no habría sido posible su realización.

También me gustaría mostrar mi agradecimiento a la familia, amigos, profesores y compañeros de máster que me han apoyado durante el curso.



## Índice general

---

<b>1</b>	<b>Introducción</b>	<b>1</b>
1.1	Contexto	1
1.2	Motivación	4
1.3	Objetivos	4
1.3.1	Objetivo principal	4
1.3.2	Objetivos secundarios	4
1.3.3	Objetivos en el Máster	5
1.4	Estructura de la memoria	5
<b>2</b>	<b>Estado del arte</b>	<b>7</b>
2.1	Sistemas de recomendación	7
2.1.1	Tipos de sistemas de recomendación	8
2.2	Funcionalidades para descubrir contenido musical	10
2.2.1	Radios	10
2.2.2	Listas de éxitos	10
2.2.3	Playlists	10

2.3	Proyectos similares . . . . .	11
2.3.1	MagicPlaylist . . . . .	11
2.3.2	Playlist Generator . . . . .	11
2.3.3	Spotalike . . . . .	12
<b>3</b>	<b>Metodología y desarrollo . . . . .</b>	<b>13</b>
3.1	Metodología. . . . .	13
3.1.1	CRISP-DM . . . . .	13
3.1.2	Scrum . . . . .	15
3.1.3	Integración de CRISP-DM en Scrum . . . . .	18
3.1.4	Aplicación de Scrum . . . . .	20
3.2	Desarrollo. . . . .	21
3.2.1	Lenguajes de programación . . . . .	21
3.2.2	Servicios en la nube . . . . .	22
3.2.3	Herramientas de desarrollo . . . . .	22
3.2.4	Bibliotecas . . . . .	23
3.2.5	Hardware empleado . . . . .	23
<b>4</b>	<b>Modelo de datos para la recomendación de playlists . . . . .</b>	<b>25</b>
4.1	Infraestructura . . . . .	25
4.2	Conjunto de datos . . . . .	27
4.3	Preprocesamiento de los datos . . . . .	27
4.4	Obtención de datos adicionales. . . . .	28
4.4.1	Creación de etiquetas . . . . .	28
4.4.2	Obtención del género musical de los artistas . . . . .	31
4.4.3	Obtención de las fechas de lanzamiento de los álbumes . . . . .	31



4.5	Selección del modelo: LightFM . . . . .	31
4.5.1	Conversión del conjunto al formato admitido . . . . .	33
4.5.2	Definición y entrenamiento del modelo . . . . .	33
4.6	Creación del punto de conexión al modelo . . . . .	34
<b>5</b>	<b>Servicio de recomendación de playlists . . . . .</b>	<b>37</b>
5.1	Informática sin servidor . . . . .	37
5.2	Infraestructura . . . . .	38
5.3	Base de datos . . . . .	39
5.4	Aplicación de funciones . . . . .	40
5.5	API REST . . . . .	43
5.6	Aplicación web . . . . .	45
<b>6</b>	<b>DevOps . . . . .</b>	<b>47</b>
6.1	Introducción . . . . .	47
6.1.1	¿Qué es DevOps? . . . . .	47
6.1.2	¿Por qué es tan importante? . . . . .	47
6.1.3	Ventajas . . . . .	48
6.1.4	Etapas del ciclo de DevOps . . . . .	48
6.1.5	Prácticas de DevOps . . . . .	49
6.2	Azure DevOps . . . . .	50
6.3	Integración continua . . . . .	50
6.4	Entrega continua . . . . .	51
6.5	Infraestructura como servicio . . . . .	52

<b>7 Experimentos y resultados.....</b>	<b>55</b>
7.1 Experimentos.....	55
7.1.1 Recomendaciones sobre playlists del conjunto de test . . . . .	55
7.1.2 Arranque en frío . . . . .	58
7.1.3 Obtención de etiquetas similares . . . . .	60
7.2 Resultados .....	60
7.2.1 Definición de métricas . . . . .	60
7.2.2 Métricas obtenidas . . . . .	62
<b>8 Conclusiones y propuestas.....</b>	<b>63</b>
8.1 Conclusiones .....	63
8.2 Propuestas .....	64
8.2.1 Funcionalidad del sistema . . . . .	64
8.2.2 Líneas de investigación . . . . .	65
<b>A Contenido del CD.....</b>	<b>67</b>
<b>Referencias bibliográficas.....</b>	<b>74</b>

## Índice de figuras

---

2.1	Comparación gráfica entre filtrado por contenido y filtrado colaborativo . . .	9
2.2	Esquema del funcionamiento de un sistema de recomendación híbrido . . .	9
2.3	Captura de la página web del proyecto MagicPlaylist . . . . .	11
2.4	Captura de la página web del proyecto Playlist Generator . . . . .	12
2.5	Captura de la página web del proyecto Spotalike . . . . .	12
3.1	Fases del modelo de referencia . . . . .	14
3.2	Marco de trabajo Scrum . . . . .	19
3.3	Ejemplo de un proceso híbrido CRISP-DM + Scrum para la entrega de proyectos de ciencia de datos . . . . .	20
4.1	Infraestructura para los procesos de aprendizaje automático en la nube . . .	26
4.2	Muestra del diccionario de traducción de emoticonos . . . . .	29
5.1	Infraestructura del servicio de recomendación de playlists . . . . .	39
5.2	Modelos admitidos por Cosmos DB . . . . .	40
5.3	Arquitectura de la base de datos del sistema de recomendación . . . . .	41
5.4	Esquema de funcionamiento de una Azure Function . . . . .	42
5.5	Esquema de funcionamiento de una API REST . . . . .	43
5.6	Página principal de la aplicación web . . . . .	44

5.7	Opción "Mis playlists" de la aplicación web . . . . .	45
5.8	Captura de pantalla de la opción "¡Voy a tener suerte!" . . . . .	46
6.1	Diagrama del ciclo de vida de Terraform . . . . .	53
8.1	Ejemplo de modelo secuencial . . . . .	65

## Índice de tablas

---

3.1	Especificaciones hardware del equipo principal . . . . .	24
3.2	Especificaciones de la instancia de cómputo . . . . .	24
7.1	Recomendación de la playlist <i>spanish playlist</i> . . . . .	56
7.2	Contenido de la playlist <i>throwback songs</i> . . . . .	57
7.3	Recomendación sobre la playlist <i>throwback songs</i> . . . . .	57
7.4	Contenido de la playlist #1013678 . . . . .	57
7.5	Recomendación sobre la playlist #1013678 . . . . .	58
7.6	Recomendación de playlist para el título "The Disney Playlist" . . . . .	59
7.7	Recomendación de playlist para el título "Christmas 2021" . . . . .	59
7.8	Resultados obtenidos en la competición de AICrowd . . . . .	62



## Índice de listados de código

---

4.1	Definición del modelo LightFM . . . . .	33
-----	---	----





# 1. Introducción

---

## 1.1. Contexto

En la actualidad, los sistemas de recomendación musical o *MRS* (*Music Recommender Systems*) han adquirido una gran importancia gracias a la aparición de los servicios de streaming musical (*Spotify*, *YouTube Music*, *Amazon Music*, ...). Mediante estos servicios, los usuarios tienen a su disposición casi todo el catálogo musical existente. Aunque gracias a los *MRS* los usuarios pueden encontrar música acorde a sus gustos, todavía existen grandes desafíos en los que se requiere investigar [1].

El interés por la investigación en estos sistemas ha experimentado un gran aumento, tanto en el mundo académico como en la industria [2]. Aunque se requiere filtrar gran cantidad de contenido musical, los *MRS* suelen tener mucho éxito a la hora de sugerir canciones que se ajustan a las preferencias de sus usuarios. Sin embargo, estos sistemas todavía están lejos de ser perfectos y con frecuencia producen recomendaciones insatisfactorias. Esto se debe en parte al hecho de que los gustos de los usuarios y las necesidades musicales dependen de una multitud de factores que no se consideran con suficiente profundidad en los enfoques actuales de *MRS*. Generalmente se centran en el concepto de las interacciones entre el usuario y el elemento (canción o playlist) o, a veces, emplean también descriptores de elementos basados en el contenido. Por el contrario, satisfacer las necesidades de entretenimiento musical de los usuarios requiere tener en cuenta aspectos intrínsecos, extrínsecos y contextuales de los oyentes [3], así como información de interacción más detallada. Por ejemplo, es sabido que la personalidad y el estado emocional de los oyentes (intrínseco) [4] [5], así como su actividad (extrínseca) [6] [7], influyen en los gustos y necesidades musicales en cada momento. También lo son los factores contextuales de los usuarios [3] [8], incluidas las condiciones climáticas, el entorno social o los lugares de interés. Además, la composición y la anotación de una lista de reproducción de música (playlists) o una sesión de escucha revelan información sobre qué canciones van bien juntas o son adecuadas para una determinada ocasión [9] [10]. Por lo tanto, los investigadores y diseñadores de *MRS*

---

deben reconsiderar a sus usuarios de manera holística para construir sistemas a la medida de las peculiaridades de cada usuario.

También resulta importante resaltar los aspectos principales que hacen que la recomendación de música requiera de un esfuerzo particular, que contrasta con la recomendación de otros elementos, como películas, libros o productos. Estos aspectos son [1]:

1. **Duración de los elementos:** En la recomendación de películas, los elementos de interés tienen una duración de 90 minutos o más. En la recomendación de libros, el tiempo de consumo es, en general, mucho mayor. Por el contrario, la duración de los elementos musicales suele oscilar entre 3 y 5 minutos (con excepciones en la música clásica).
2. **Magnitud de los elementos:** El tamaño de los catálogos de música comercial se encuentra en el rango de decenas de millones de pistas musicales, mientras que los servicios de transmisión de películas tienen que lidiar con tamaños de catálogo mucho más pequeños, generalmente de decenas de miles de películas y series. Por lo tanto, la escalabilidad es un tema mucho más importante en la recomendación de música que en la recomendación de películas.
3. **Consumo secuencial:** A diferencia de las películas, las pistas musicales se consumen con mayor frecuencia de forma secuencial, más de una a la vez (en una sesión de escucha o en una lista de reproducción). Esto genera una serie de desafíos para un MRS, que se relacionan con la identificación de la disposición correcta de elementos en una lista de recomendaciones.
4. **Recomendación de elementos recomendados anteriormente:** El usuario de un MRS puede admitir la recomendación de la misma pista musical nuevamente, en un momento posterior, a diferencia de un sistema de recomendación de películas o productos, donde generalmente no se incluyen recomendaciones repetidas.
5. **Comportamiento de consumo:** La música, a menudo, se consume de forma pasiva, de fondo. Si bien esto no es un problema en sí, puede afectar a la obtención de preferencias. En particular, cuando se utiliza la retroalimentación implícita para inferir las preferencias del oyente, el hecho de que un oyente no esté prestando atención a la música (puede ocurrir, por ejemplo, que no se salte una canción) podría interpretarse erróneamente como una señal positiva.
6. **Intención y propósito de escucha:** La música tiene varios propósitos para las personas y, por lo tanto, condiciona la forma a su intención de escucharla. De todos los propósitos que pueden darse, destacan tres: la autoconciencia (relación privada con la escucha de música), la relación social (describe el uso de la música para sentirse cerca de los amigos y para expresar identidad y valores a los demás) y regulación del estado de ánimo (uso de las emociones).
7. **Emociones:** Se sabe que la música evoca emociones muy fuertes. Sin embargo, esta es una relación mutua, ya que también las emociones de los usuarios afectan las preferencias musicales. Debido a esta fuerte relación entre la música y las emociones, el

problema de describir automáticamente la música en términos de palabras emocionales es un área de investigación activa, comúnmente conocida como reconocimiento de emociones musicales o *MER* (*Music Emotion Recognition*).

8. **Contexto de escucha:** Los aspectos situacionales o contextuales tienen una fuerte influencia en la preferencia musical, el consumo y el comportamiento de interacción. Por ejemplo, es probable que una persona escuche una playlist diferente cuando se prepara para una cena romántica que cuando se prepara para salir con amigos un viernes noche. Los tipos de contexto considerados con más frecuencia incluyen la ubicación y el momento, e incluso el clima.

Para finalizar esta introducción, cabe señalar que actualmente existen algunos retos que destacan en todas las líneas de investigación dentro del campo de los sistemas de recomendación musicales [1]:

1. **Problema del arranque en frío:** Uno de los principales problemas de los sistemas de recomendación en general, y de los sistemas de recomendación de música en particular, es el problema de inicio en frío. Por ejemplo, cuando un nuevo usuario se registra en el sistema, o se agrega un nuevo artículo al catálogo, y el sistema no tiene suficientes datos asociados con estos artículos o usuarios. En tal caso, el sistema no puede recomendar correctamente elementos existentes a un nuevo usuario (problema de nuevo usuario) o recomendar un nuevo elemento a los usuarios existentes (problema de nuevo elemento).
2. **Continuación automática de playlists:** En su definición más genérica, una lista de reproducción es simplemente una secuencia de pistas destinadas a ser escuchadas juntas. La tarea de generación automática de listas de reproducción o *APG* (*Automatic Playlist Generation*) se refiere a la creación automática de estas secuencias de pistas. En este contexto, el orden de las canciones para generar una lista de reproducción aparece como una característica de las *APG*, que requiere de un esfuerzo muy complejo.
3. **Evaluación de sistemas de recomendación musical:** Teniendo sus raíces en el aprendizaje automático y la recuperación de información, los sistemas de recomendación originalmente adoptaron métricas de evaluación de estos campos. De hecho, el *accuracy* y las medidas cuantitativas relacionadas (*precision*, *recall* o medidas de error), siguen siendo los criterios más comunes que se emplean para juzgar la calidad de las recomendaciones de un sistema. Además, en los últimos años han surgido nuevas medidas que se adaptan al problema de las recomendaciones. Estas medidas abordan las particularidades de los sistemas de recomendación y miden, por ejemplo, la utilidad, la novedad o la causalidad de un artículo. Sin embargo, un problema importante con este tipo de medidas es que integran factores que son difíciles de describir matemáticamente. Por esta razón, a veces existe una variedad de definiciones diferentes para cuantificar el mismo aspecto más allá de la precisión.

---

## 1.2. Motivación

La principal motivación de este TFM surge por el interés de seguir investigando en el campo de los sistemas de recomendación y continuar con el trabajo realizado en el proyecto *Generación automática de playlist de canciones mediante técnicas de minería de datos* [11] de tal forma que podamos mejorar el rendimiento del modelo obtenido y, haciendo uso de este, montar un sistema de recomendación de listas de reproducción musicales que sea empleado por una aplicación web. Para llegar a este objetivo hemos empleado *DevOps*, un conjunto de prácticas de las cuales hablaremos en el Capítulo 6.

## 1.3. Objetivos

En esta sección vamos a explicar los principales objetivos que queremos lograr durante la elaboración de este trabajo.

### 1.3.1. Objetivo principal

El principal objetivo de este trabajo es construir un sistema de recomendación de playlists de canciones. Nuestro sistema también deberá ser capaz de solventar el problema del *arranque en frío*, cuando la playlist sobre la que realizar recomendaciones no se encuentra en nuestro conjunto de datos (incluso buscando playlists similares), o cuando únicamente se le proporciona un título a nuestro sistema.

### 1.3.2. Objetivos secundarios

Aparte del objetivo principal que hemos mencionado anteriormente, durante la realización de este trabajo también nos proponemos alcanzar los siguientes objetivos:

- Empleo de la metodología *Scrum* junto a prácticas de *DevOps*.
- Despliegue de la infraestructura necesaria en un proveedor de servicios en la nube.
- Diseño de un modelo capaz de realizar recomendaciones sobre playlists.
- Llevar a producción nuestro modelo de recomendación, creando un *endpoint* capaz de realizar las predicciones frente a las consultas recibidas.
- Creación de una aplicación web que haga uso del sistema de recomendación.
- Estudio de los resultados obtenidos.

### 1.3.3. Objetivos en el Máster

Uno de los objetivos del *Máster Universitario en Ingeniería Informática* es, independientemente de la especialización recibida por el alumno durante el Grado, reconciliar sus conocimientos con el resto de las intensificaciones, aportándole así un valor añadido que le habilitará para trabajar en cualquier tipo de proyecto [12].

En nuestro proyecto hemos abarcado las intensificaciones pertenecientes al *Grado en Ingeniería Informática* de la siguiente manera:

- **Computación:** Análisis de datos y desarrollo de un modelo de recomendación.
- **Ingeniería del Software:** Aplicación de las metodología *Scrum* y *DevOps*.
- **Tecnologías de la Información:** Diseño e implementación de una aplicación web.
- **Ingeniería de Computadores:** Uso de servicios e infraestructura en la nube.

### 1.4. Estructura de la memoria

Hemos estructurado la memoria en los siguientes capítulos:

- **Capítulo 1 – Introducción:** Contiene información acerca del tema elegido y la estructura de este trabajo, así como los motivos que han llevado a su elección.
- **Capítulo 2 – Estado del arte:** En este capítulo hablaremos de los sistemas de recomendación, los tipos que existen, y de las funcionalidades que ofrecen los servicios de streaming para descubrir contenido musical. También mencionaremos proyectos similares que existen en la actualidad.
- **Capítulo 3 – Metodología y desarrollo:** Explicación de la metodología utilizada para realizar el trabajo junto a las herramientas software y hardware empleadas.
- **Capítulo 4 – Modelo de datos para la recomendación de playlists:** Contiene toda la información referente al proyecto de ciencia de datos (estudio del conjunto de datos empleado, preprocesamiento de datos, definición y entrenamiento del modelo, etc.).
- **Capítulo 5 – Servicio de recomendación de playlists:** Detalle del desarrollo del servicio de recomendación, desde la infraestructura necesaria y los servicios empleados, hasta su puesta en funcionamiento.
- **Capítulo 6 – DevOps:** Descripción de la metodología, técnicas y herramientas *DevOps* que hemos empleado en la realización de este trabajo.
- **Capítulo 7 – Experimentos y resultados:** Detalle de los experimentos realizados para comprobar las recomendaciones de nuestro sistema y resultados obtenidos al evaluar nuestro sistema.

- 
- **Capítulo 8 – Conclusiones y propuestas:** Contiene las conclusiones que se han obtenido tras concluir el trabajo junto a una serie de propuestas que se desean realizar en un futuro.

## 2. Estado del arte

---

En este capítulo describimos en qué consiste un sistema de recomendación y los distintos tipos de sistemas que existen. También hablaremos de algunas de las funcionalidades que permiten al usuario descubrir nuevas canciones en los servicios de streaming musical más populares. Por último, haremos un breve repaso a proyectos similares.

### 2.1. Sistemas de recomendación

Cuando hablamos de sistemas de recomendación, nos referimos a aquellos sistemas que son capaces de predecir el grado de preferencia que un usuario tendrá para un conjunto de ítems, recomendándole aquellos que resulten de más interés para él.

En la actualidad, los sistemas de recomendación adquieren un papel fundamental, ya que gracias a Internet las personas disponen de muchas opciones para elegir. Por ejemplo, en un videoclub nos encontramos un número limitado de películas y que depende tamaño del establecimiento. Por el contrario, si nos fijamos en *Netflix*, existe un número elevado de películas que podemos ver en línea. Con esto surge el problema de que las personas, ante tal cantidad de contenido, tienen dificultades para seleccionar y visualizar aquellas películas que pueden ser de su interés. En este escenario es donde los sistemas de recomendación ayudan a los usuarios a filtrar el contenido disponible de acuerdo con sus intereses y/o visualizaciones anteriores [13].

En un sistema de recomendación, los datos de los que hacemos uso pueden ser de dos tipos [14]:

- **Información característica:** Información sobre los ítems a recomendar (palabras clave, categorías, ...) y usuarios (preferencias, perfiles, ...).

- 
- **Interacciones usuario-ítem:** Información sobre calificaciones, número de elementos adquiridos (compras), “likes”, etc.

### 2.1.1. Tipos de sistemas de recomendación

Existen tres tipos importantes de sistemas de recomendación: los de *filtrado colaborativo*, los de *filtrado basado en contenido* y los *sistemas de recomendación híbridos*.

#### Filtrado colaborativo

Este método se basa en recopilar y analizar información relacionada con los comportamientos del usuario, como actividades, preferencias, etc., y realizar una predicción de lo que le puede gustar en función de su similaridad con otros usuarios [14].

- Ejemplo: Si un usuario **A** valora positivamente las películas *John Wick*, *Misión Imposible* y *Jungla de Cristal*, y un usuario **B** valora positivamente las películas *Misión Imposible*, *Jungla de Cristal* y *El Caso Bourne*, entonces tienen intereses similares y al usuario **A** podría gustarle la película *El Caso Bourne* y al usuario **B** podría gustarle la película *John Wick*.

#### Basado en contenido

En este tipo de filtrado, nos centramos en las características o atributos de los ítems. En los sistemas de recomendación basados en contenido, las palabras clave se usan para describir los elementos. Además, también se crea un perfil de usuario para indicar el tipo de ítem que podría gustarle [14].

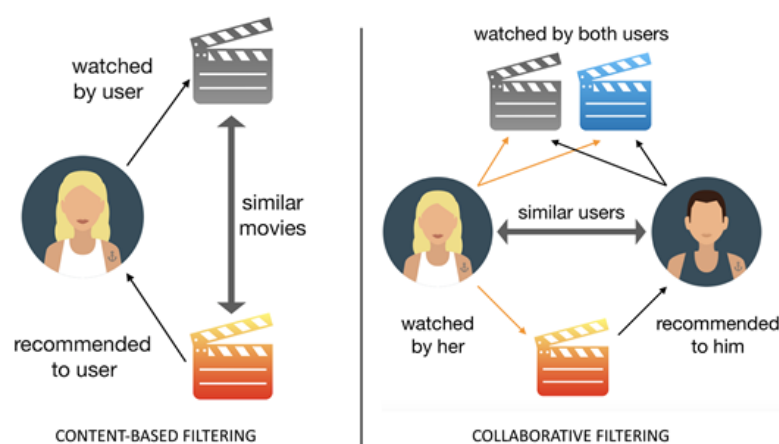
- Ejemplo: Si un usuario ha valorado positivamente *Star Trek*, *Matrix* y *Blade Runner*, sabemos que le gustan las películas cuyo género es *ciencia ficción*. Por lo tanto, las recomendaciones se adaptarán para mostrar películas de dicho género.

En la Figura 2.1 podemos ver de forma gráfica una comparación de cómo funcionan los sistemas de recomendación basados en filtrado colaborativo y los sistemas de recomendación basados en contenido.

#### Sistemas de recomendación híbridos

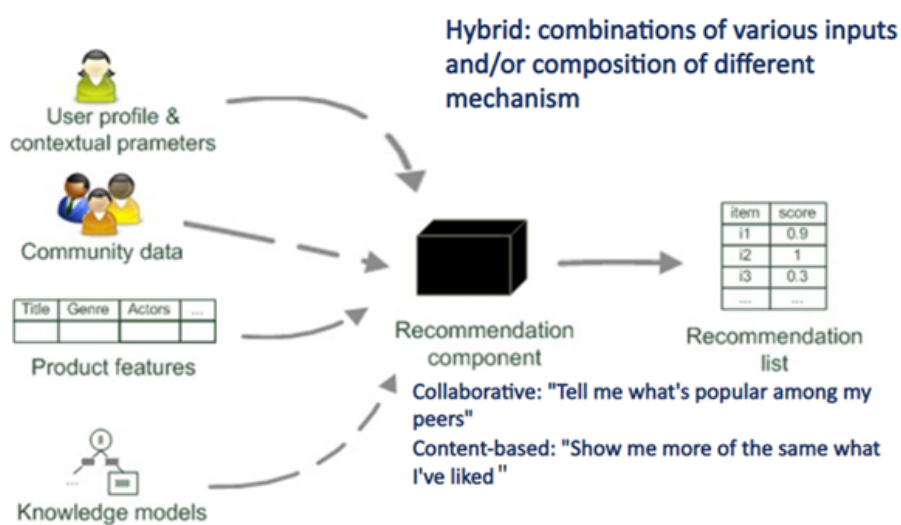
Se basa en combinar los métodos de filtrado colaborativo y filtrado basado en contenido. Investigaciones recientes muestran que combinar ambos tipos de sistemas es más efectivo y se proporcionan recomendaciones más precisas [14].





**Figura 2.1:** Comparación gráfica entre filtrado por contenido y filtrado colaborativo [15]

En la Figura 2.2 podemos ver de forma gráfica cómo funciona un sistema de recomendación híbrido.



**Figura 2.2:** Esquema del funcionamiento de un sistema de recomendación híbrido [16]

---

## 2.2. Funcionalidades para descubrir contenido musical

Los servicios como *Spotify*, *Apple Music* y *Amazon Music*, que son algunos de los más destacados, ofrecen funcionalidades muy similares entre ellos. Si nos fijamos en aquellas que el usuario utiliza para añadir contenido a su colección, las más empleadas son las siguientes: radios, listas de éxitos y playlists definidas por el servicio (proveedor).

### 2.2.1. Radios

Las radios, también conocidas como emisoras, son una herramienta que se basa en que cuando un usuario está reproduciendo una canción, un álbum o un artista, al iniciar este servicio se reproducen canciones relacionadas/similares al tipo de contenido elegido de forma aleatoria (varía en cada ejecución). Por ejemplo, si un usuario inicia la radio de la cantante *Jennifer López*, se reproducen canciones de esta artista junto a canciones de cantantes que son similares a ella. Otro ejemplo, en el caso de elegir una canción, la radio reproducirá canciones que sean similares o estén relacionadas con la que hemos indicado (ya sea por el género musical, época, ritmo, artista similar, ...).

Algunos de los proveedores de música en streaming también crean sus radios, pero en este caso pueden ser de una temática establecida

### 2.2.2. Listas de éxitos

Las listas de éxitos musicales llevan con nosotros varias décadas. En los servicios de música en streaming, nos ofrecen listas de éxitos relacionadas con el número de reproducciones que tienen las canciones en ese momento, ya sea a nivel mundial o por país. De igual manera que se hace con el número de reproducciones, si el servicio ofrece la posibilidad de comprar contenido, se ofrecen listas con las ventas de canciones en un momento dado. También se crean listas de éxitos respecto a estilos musicales.

### 2.2.3. Playlists

Las playlists o listas de reproducción, son una de las funcionalidades más populares en los servicios de streaming musical. Un usuario crea una lista con un título y añade una serie de canciones. Conforme se van añadiendo canciones a dicha playlist, el servicio va sugiriendo canciones que podrían estar relacionadas (ya sea por similitud entre canciones o porque se han encontrado en otras playlists con títulos parecidos). En este caso los usuarios pueden compartir sus listas con otros, aparte de que el servicio también crea múltiples playlist que pueden seguirse.

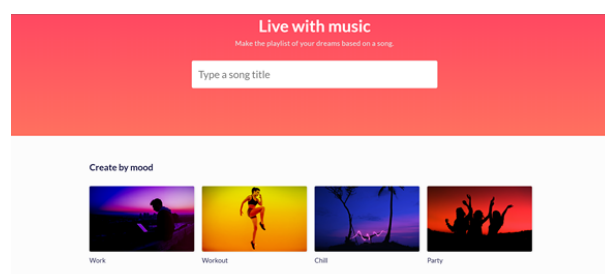
Los proveedores de contenido musical también suelen crear sus propias playlist, de diferentes temáticas, para ofrecer a sus usuarios.

## 2.3. Proyectos similares

Para finalizar este capítulo, vamos a hacer un repaso de algunos proyectos similares al que se va a tratar en este trabajo, disponibles en Internet. Antes de comenzar, cabe destacar que los proyectos que hemos localizado sólo generan una playlist a partir de un cantante o canciones indicadas (salvo en un trabajo donde se crean listas mediante estado anímico y género). No hemos encontrado proyectos que, partiendo de una playlist propia, sean capaces de recomendarnos otras, ya sean de otros usuarios o generadas, como sí lo hace nuestro sistema. Otra consideración a tener en cuenta es que todos los proyectos son para la creación de listas en *Spotify*.

### 2.3.1. MagicPlaylist

Este proyecto ofrece la posibilidad de crear una playlist a partir de una canción indicada o bien mediante el estado anímico junto a un género musical. Una vez que la lista de reproducción ha sido creada, nos ofrece la oportunidad de guardarla en *Spotify* [17].



**Figura 2.3:** Captura de la página web del proyecto MagicPlaylist

### 2.3.2. Playlist Generator

Con este generador, a partir de una canción inicial y una final elegidas por un usuario, junto al número de canciones que se desea, se genera una playlist. La forma en la que se genera es la siguiente: se crea un grafo donde los nodos son las canciones incluidas en el conjunto de datos del proyecto y se intenta encontrar un camino entre las canciones que hemos indicado. En este caso también podemos importarla a *Spotify* [18].

### Generate Your Playlist

To generate your playlist, you must select the first and last song of the playlist and set the desired number of songs.

#### First Song

Artist: Taylor Swift

Select Artist

Song: Shake It Off

Select Song

#### Last Song

Artist: Katy Perry

Select Artist

Song: Hot N Cold

Select Song

Number of songs in the playlist:

15

Create Playlist

#### Generated Playlist:

1. Shake It Off by Taylor Swift
2. This Is How We Do by Katy Perry
3. Chasing The Sun by The Wanted
4. Animals by Maroon 5
5. Beauty And A Beat by Justin Bieber
6. Want U Back by Cher Lloyd
7. Rude Boy by Rihanna
8. Party In The U.S.A. by Miley Cyrus
9. Sweet Dreams by Beyoncé
10. Hot N Cold by Katy Perry

**Figura 2.4:** Captura de la página web del proyecto Playlist Generator

### 2.3.3. Spotalike





Este último proyecto, en fase beta, aparte de usar las herramientas ofrecidas por *Spotify* también hace uso de las disponibles por el portal de música *Last.fm*. Su formato es similar al de un buscador: se introduce el nombre de una canción y el sistema muestra los resultados (canciones en nuestro caso) que ha encontrado [19]. En la Figura 2.5 se muestra un ejemplo de ejecución.

SONGS SIMILAR TO:

## ...Baby One More Time

by Britney Spears

[Add to Spotify](#) 3 hours 3 minutes of similar songs. Enjoy.

TRACK	ARTIST	DURATION
 ...Baby One More Time	Britney Spears	3:31
 Stronger	Britney Spears	3:23
 Oops!...I Did It Again	Britney Spears	3:31
 Genie in a Bottle	Christina Aguilera	3:37

**Figura 2.5:** Captura de la página web del proyecto Spotalike

## 3. Metodología y desarrollo

---

### 3.1. Metodología

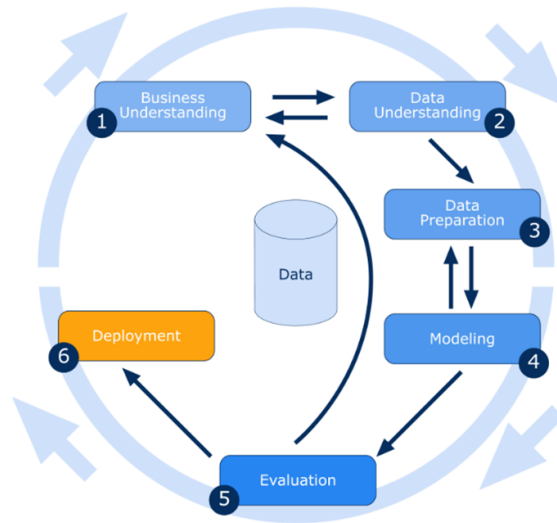
En este proyecto se desarrolla un modelo de aprendizaje automático que recomienda playlists, junto a un servicio de recomendación y una aplicación web. Por lo tanto, vamos a tener dos trabajos diferenciados, el de ciencia de datos (creación del modelo) y el de desarrollo de software convencional (aplicación web que hace uso de una *API REST*).

Para ello, combinamos la metodología *CRISP-DM*, que se emplea en minería de datos, junto con *SCRUM*, marco de trabajo empleado en el desarrollo de software.

#### 3.1.1. CRISP-DM

El modelo *CRISP-DM* (*Cross Industry Standard Process for Data Mining*) provee de una descripción normalizada del ciclo de vida de un proyecto de minería de datos [20]. Se desarrolla en 6 pasos, los cuales explicamos a continuación y podemos ver de forma gráfica en la Figura 3.1:

1. **Comprensión del negocio:** Nos centramos en comprender los objetivos y requisitos del proyecto desde una perspectiva comercial, para luego convertir este conocimiento en una definición de problema de minería de datos y un plan preliminar [22] [23].
2. **Estudio y comprensión de los datos:** Comenzamos con una recopilación de datos inicial y continuamos con las actividades para comprenderlos, identificar problemas de calidad, descubrir las primeras ideas o detectar subconjuntos interesantes para formar hipótesis de información oculta [20].



**Figura 3.1:** Fases del modelo de referencia CRISP-DM [21]

3. **Preparación de los datos:** La fase de preparación cubre todas las actividades para construir el conjunto final de datos a partir de los datos iniciales que se encuentran sin procesar [20].
4. **Modelado:** Se seleccionan y aplican técnicas de modelado. Dado que algunas técnicas, como las redes neuronales, tienen requisitos específicos con respecto a la forma de los datos, podemos encontrarnos el caso de tener que volver a la fase de preparación de datos [20].
5. **Evaluación:** Una vez que se han construido uno o varios modelos que parecen tener una alta calidad en base a las funciones de pérdida que se hayan seleccionado, estos deben ser probados para garantizar que generalicen bien al usar datos nuevos, no usados en el entrenamiento, y que todos los problemas comerciales clave se hayan considerado suficientemente. El resultado final es la selección de los modelos con mejores resultados [20].
6. **Despliegue:** En general, consiste en desplegar una representación de código del modelo en un sistema operativo para calificar o categorizar nuevos datos invisibles, a medida que surjan, y crear un mecanismo para el uso de esa nueva información en la solución del problema comercial inicial. Es importante destacar que la representación del código también debe incluir todos los pasos de preparación de datos que conducen al modelado, para que el modelo trate los nuevos datos sin procesar para ser usados por el modelo [23].

### 3.1.2. Scrum

*Scrum* es un marco de trabajo para el desarrollo ágil de proyectos, en principio surgido en la industria del software, pero de suficiente sencillez y flexibilidad como para ser aplicado en contextos muy diversos. Dentro de este marco, el proceso de desarrollo de un proyecto se concibe como una sucesión de ciclos cortos de trabajo denominados *sprints*, obteniendo de cada uno de ellos un producto funcional que va completándose en forma iterativa [24] [25].

#### Valores

Para trabajar con una metodología Scrum, se necesita una base firme de valores que sirvan como fundamento para el proceso y los principios del equipo. A través del uso del trabajo en equipo y la mejora continua, Scrum tanto crea como depende de los siguientes valores [26]:

- **Coraje:** Porque no estamos solos, nos sentimos apoyados y tenemos más recursos a nuestra disposición. Esto nos da el coraje para enfrentar desafíos más grandes [26].
- **Foco:** Porque nos enfocamos en sólo unas pocas cosas a la vez, trabajamos bien juntos y producimos un resultado excelente. De este modo logramos entregar ítems valiosos antes [26].
- **Compromiso:** Porque tenemos gran control sobre nuestro destino, nos comprometemos más al éxito [26].
- **Respeto:** A medida que trabajamos juntos, compartiendo éxitos y fracasos, llegamos a respetarnos los unos a los otros, y a ayudarnos mutuamente a convertirnos en merecedores de respeto [26].
- **Apertura:** Durante el trabajo en conjunto expresamos cotidianamente cómo nos va y qué problemas encontramos. Aprendemos que es bueno manifestar las preocupaciones, para que éstas puedan ser tomadas en cuenta [26].

#### El equipo Scrum

Un equipo de gestión de proyectos Scrum consta de tres roles. Los roles son autoorganizados y multifuncionales, y están diseñados para maximizar la flexibilidad, la creatividad y la productividad. Dichos roles son los siguientes:

- **Product Owner:** Es responsable de maximizar el valor del producto resultante del trabajo del equipo Scrum. La forma en que se hace esto puede variar ampliamente entre organizaciones, equipos Scrum e individuos [27].

- 
- **Development Team:** El equipo de desarrollo es un grupo de profesionales, como desarrolladores, programadores o diseñadores, que entregan el producto y crean el *incremento* (la funcionalidad del producto de trabajo que se presenta a las partes interesadas al final del sprint y que potencialmente podría entregarse al cliente) [27].
  - **Scrum Master:** Es el responsable de asegurar que el marco de trabajo Scrum es entendido y adoptado, asegurándose de que el equipo Scrum trabaja ajustándose a la teoría, prácticas y reglas de Scrum. Está al servicio del equipo Scrum, ayuda a las personas externas al equipo a entender qué interacciones con el equipo pueden ser de ayuda y cuáles no. Ayuda a todos a modificar estas interacciones para maximizar el valor creado por el equipo Scrum [28].

## Eventos Scrum

La gestión de proyectos Scrum se aplica siguiendo un flujo de trabajo compuesto por eventos. Los eventos están encuadrados en el tiempo, lo que significa que tienen un tiempo de duración máximo que no se puede exceder. El objetivo de esto es reducir el tiempo perdido durante el proceso de desarrollo, así como permitir que se produzca una transparencia e inspección críticas. Los eventos son [27]:

- **Sprint:** Son la esencia misma de la gestión de proyectos Scrum. Están encuadrados en el tiempo, son iterativos y tienen el propósito de lograr un objetivo, generalmente para crear un producto utilizable y potencialmente liberable o incluso para mover un elemento de *En ejecución* a *Listo*. Sus marcos de tiempo, que generalmente son de dos semanas, son consistentes durante todo el proceso de desarrollo, con un nuevo sprint que comienza inmediatamente después de la conclusión del anterior [27].
- **Sprint Planning:** La planificación de sprints es el evento que ocurre al comienzo de cada sprint, donde todo el equipo de gestión de proyectos de Scrum se reúne para planificar el próximo sprint. Se realizan discusiones y arreglos para responder a dos preguntas principales: *¿Qué se puede lograr al final de este próximo sprint?* y *¿Cómo vamos a lograrlo y cuáles son las acciones necesarias para lograrlo?*. Una vez que se responda a estas preguntas, el equipo de gestión de proyectos de Scrum tendrá una idea del alcance del trabajo que se espera durante el próximo sprint y el tiempo asignado para cada tarea. También es durante la planificación del sprint cuando se organiza el *Sprint Backlog*, que es una lista de tareas que reflejan el trabajo que se debe completar [27].
- **Daily Scrum:** Es una reunión de 15 minutos que se lleva a cabo a la misma hora todos los días del sprint. Es donde el equipo de desarrollo discute sus logros del día anterior y sus expectativas para el siguiente. Aunque está organizado por el Scrum Master, la reunión es únicamente del equipo de desarrollo. El propósito del Daily Scrum es sincronizar la actividad, mejorar la colaboración e identificar cualquier obstáculo.
- **Sprint Review:** La revisión de sprint es una reunión informal que se lleva a cabo al final de cada sprint, donde el equipo Scrum presenta su incremento a las partes interesa-



das. Durante la revisión, todos comparten sus comentarios y colaboran en lo que se debe hacer en el próximo sprint para optimizar el valor del producto. La revisión termina con un Product Backlog revisado que describe los elementos para el próximo sprint según lo que se discutió y acordó colectivamente [27].

- **Sprint Retrospective:** La retrospectiva ocurre después de la revisión y antes de la siguiente planificación del sprint. Es una reunión donde el Scrum Team reflexiona sobre los procedimientos del sprint anterior y establece espacios de mejora en el siguiente sprint. Aunque el objetivo de la gestión de proyectos Scrum es permitir que se produzcan mejoras en cualquier momento, el *Sprint Review* es una oportunidad para que el equipo se centre en la inspección y la adaptación formalmente [27].

## Artefactos Scrum

Los artefactos pueden verse como las herramientas necesarias para entregar y completar el trabajo. Estos son los principales artefactos en la gestión de proyectos Scrum [27]:

- **Product Backlog:** Gestionado por el Product Owner, el Product Backlog es donde se enumeran todos los requisitos necesarios para un producto viable en orden de prioridad. Este artefacto evoluciona constantemente con el producto y el equipo, pero nunca está completo. Incluye todas las características, funciones, requisitos, mejoras y correcciones que autorizan cualquier cambio que se realice en el producto en versiones futuras [27].
- **Sprint Backlog:** Enumera las tareas y los requisitos que el equipo debe cumplir durante el próximo sprint. Es un activo del equipo de desarrollo, que a veces va acompañado de un tablero de tareas de Scrum. Dicho tablero se emplea para visualizar el progreso de las tareas en el sprint actual y cualquier cambio que se realice, en un formato de *Tareas pendientes, en ejecución y listo* (también conocido como *ToDo*) [27].
- **Incremento:** El incremento de producto es el resultado de todos los elementos del Product Backlog logrados durante un sprint. Es la funcionalidad del producto de trabajo que se presenta a las partes interesadas al final del sprint, y se considera un paso hacia el objetivo final [27].

## Marco de trabajo Scrum

Los proyectos que emplean Scrum comienzan con una visión clara, que podría cambiar, y un conjunto de características del producto ordenadas por importancia. Como hemos visto, estas características son parte del Product Backlog, siendo mantenidas por el cliente o el Product Owner.

Para comenzar a trabajar, el equipo selecciona tareas del Product Backlog que se pueden completar en el sprint, creando el *Sprint Backlog*, que consta de las características y tareas acordadas en el *Sprint Planning* y que se llevarán a cabo durante el sprint.

---

Una vez que el equipo se ha comprometido con el *Sprint Backlog*, comienza el trabajo. El equipo está protegido de interrupciones y se le permite concentrarse en cumplir el objetivo marcado para esta iteración.

Durante el sprint, el equipo se reúne diariamente en el *Dayly Scrum* antes de comenzar a trabajar. Los integrantes del equipo forman un círculo y cada miembro comenta lo que hizo el día anterior, lo que planean hacer y las dificultades encontradas.

Al final de cada sprint, en la retrospectiva del Sprint, el equipo se reúne y expone el trabajo que ha completado a las partes interesadas, recopilando todos los comentarios que afectarán a las tareas que se trabajarán en el próximo sprint. Esta reunión es crítica para el correcto uso de Scrum, ya que se centra en sus tres pilares fundamentales [29]:

- **Transparencia:** Requiere que los aspectos de Scrum sean definidos por un estándar común de tal modo que los observadores compartan un entendimiento común de lo que están viendo [30].
- **Inspección:** Los usuarios de Scrum deben inspeccionar frecuentemente los artefactos de Scrum y el progreso hacia un objetivo para detectar variaciones indeseadas. Su inspección no debe ser tan frecuente como para que interfiera en el trabajo [30].
- **Adaptación:** Si un inspector determina que uno o más aspectos de un proceso se desvían de límites aceptables y que el producto resultante será inaceptable, el proceso o el material que está siendo procesado deben ajustarse. Dicho ajuste debe realizarse cuanto antes para minimizar desviaciones mayores [30].

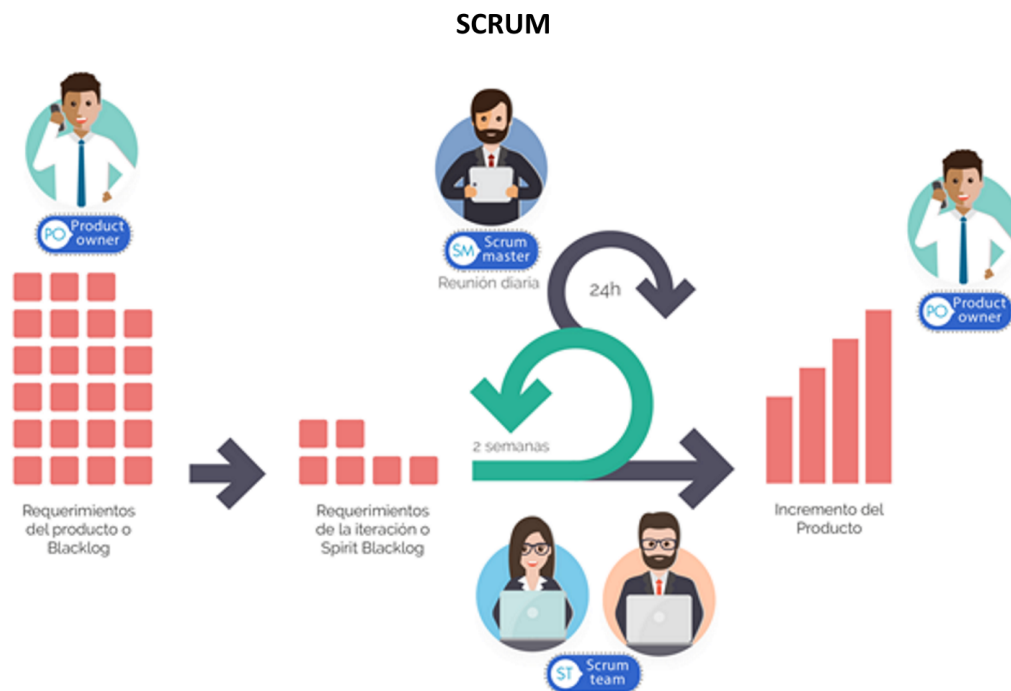
En la Figura 3.2 podemos ver de forma gráfica un resumen del desarrollo dentro del marco de trabajo de Scrum.

### 3.1.3. Integración de CRISP-DM en Scrum

Una vez que hemos repasado los principios básicos de CRISP-DM y Scrum, vamos a ver cómo se pueden emplear de forma conjunta.

Cuando integramos la metodología CRISP-DM en Scrum para la entrega del proyecto, hay determinados puntos que resultan de especial interés:

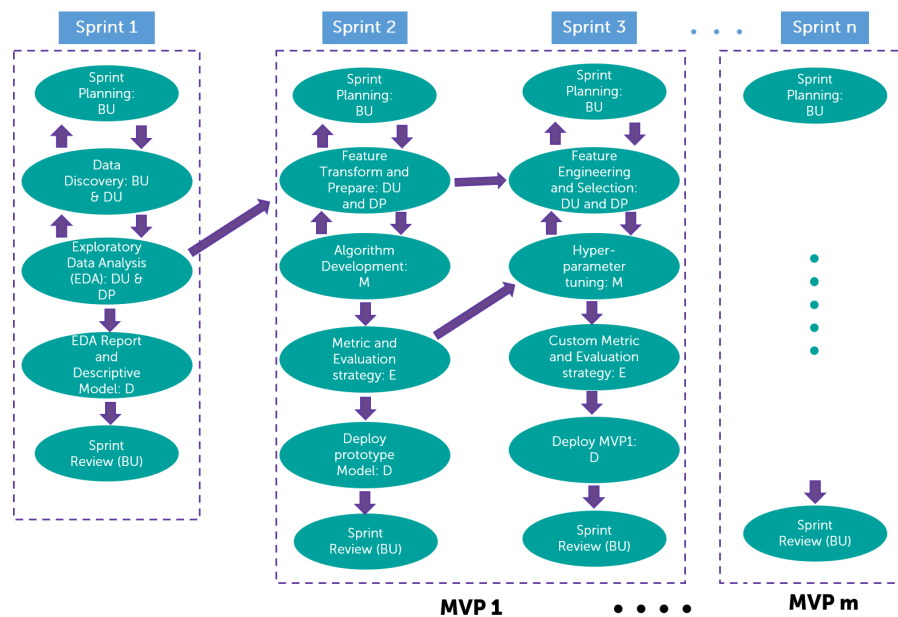
- Las reuniones de planificación (*Sprint Planning*) y, hasta cierto punto, las revisiones (*Sprint Reviews*) ofrecen una gran oportunidad para interactuar con las partes interesadas sobre el requisito de comprensión empresarial y de datos.
- A medida que entregamos iterativamente en sprints, nuestra comprensión comercial y de datos evoluciona, así como la madurez y viabilidad del modelo.



**Figura 3.2:** Marco de trabajo Scrum [26]

- Las primeras 3 fases del proceso CRISP-DM (compresión del negocio, estudio y comprensión de los datos y preparación de los datos) generalmente requieren un sprint debido al descubrimiento y la naturaleza exploratoria de las tareas. El resultado de estas fases puede ser un informe de análisis exploratorio de datos o un modelo descriptivo sencillo que se puede entregar como producto mínimo viable por sí solo, si todas las partes involucradas están interesadas. Podemos ver el resultado con más detalle en la Figura 3.3 [31].
- Diferentes versiones de un modelo para el mismo alcance, o varios modelos para diferentes alcances, se pueden entregar como mejora del proceso de software o un producto mínimo viable en uno o pocos sprints, dependiendo de la complejidad del trabajo y capacidad del equipo de desarrollo [31].

Como podemos ver, Scrum y CRISP-DM se complementan entre sí para la entrega eficiente y eficaz de proyectos de ciencia de datos. CRISP-DM guía los proyectos de ciencia de datos sobre qué entregar y en qué secuencia, mientras que Scrum nos ayuda a entregar un producto funcional con éxito en términos de eficiencia y satisfacción de las partes interesadas [31].



**Figura 3.3:** Ejemplo de un proceso híbrido CRISP-DM + Scrum para la entrega de proyectos de ciencia de datos [31]

### 3.1.4. Aplicación de Scrum

Para la realización de este trabajo hemos seguido la metodología *Scrum*, pero realizando una serie de modificaciones al ser un Trabajo de Fin de Máster y ser un equipo de trabajo pequeño (unipersonal).

Los roles y responsabilidades de *Scrum* se han asignado de la siguiente manera:

- **Product Owner:** Juan Ángel Aledo y José Antonio Gámez.
- **Scrum Master:** Miguel Ángel Cantero, y en ocasiones los tutores.
- **Equipo de desarrollo:** Miguel Ángel Cantero

Para los eventos que define la metodología *Scrum*, hemos tomado la decisión de que cada sprint tendrá una duración de 2 semanas. Cada sprint da comienzo con un *Sprint Planning*, el cual se ha llevado a cabo de forma presencial, cuando ha sido posible, y mediante *Microsoft Teams*. En dicha reunión ha estado presente el equipo de desarrollo, *Scrum Master* y *Product Owner* para dejar definido el *Sprint Backlog* para trabajar durante todo el sprint. La duración de este evento es de 20 a 30 minutos.

Puesto que el equipo de trabajo está inmerso en otros proyectos, externos a esta organización, no ha sido posible realizar eventos *Daily Scrum* tal y como se define en la guía

*Scrum*. En su lugar, se ha hecho una reunión a mitad de cada sprint (dependiendo de la disponibilidad de los integrantes) en un mismo horario y empleando también *Microsoft Teams*. Tales reuniones tienen el mismo fin que el *Daily Scrum*.

La *Sprint Review* y la retrospectiva han sido realizadas de forma presencial entre el *Product Owner* y *Scrum Master* al finalizar el sprint que se había llevado a cabo. En ese evento se discutían las dificultades, errores y logros alcanzados por el equipo de trabajo durante el sprint, tanto por parte del equipo de desarrollo, como por parte del *Product Owner* y el *Scrum Master*. Siguiendo las reglas establecidas en la guía *Scrum*, el *Sprint Review* y las retrospectivas siempre se han llevado a cabo el mismo día, durante el mismo horario y lugar.

En lo relativo a los artefactos de *Scrum*, el *Sprint Backlog* es definido por el *Product Owner* durante el comienzo de cada sprint de acuerdo con el equipo de trabajo. Una vez definido, se publica en los *Azure Boards* dentro del proyecto *DevOps* de nuestro equipo. En este servicio, hemos tenido toda la información del sprint actual, tareas, progreso y backlogs durante todo el desarrollo del proyecto.

Para finalizar, el *Product Backlog* que fue definido al comienzo del proyecto por el *Product Owner*, ha ido cambiando a lo largo del proyecto conforme han surgido nuevas necesidades durante la fase de desarrollo, siendo revisado y aprobado por el equipo de trabajo.

## 3.2. Desarrollo

En esta sección vamos a hacer un breve repaso de todas las tecnologías que hemos empleado para el trabajo, así como de los medios hardware donde se ha realizado y probado el sistema.

### 3.2.1. Lenguajes de programación

Para el desarrollo del modelo de aprendizaje automático, hemos empleado el lenguaje de programación *Python* (en su versión 3.8). Este lenguaje resulta muy útil en el desarrollo de aplicaciones complejas tanto numéricas como científicas. Debido a su versatilidad, *Python* es uno de los lenguajes más empleados en la actualidad. Un ejemplo de ello es que, recientemente, se ha convertido en el segundo lenguaje más usado en *GitHub* superando así al lenguaje *Java* [32].

En el campo de ciencia de datos, *Python* se ha convertido en un lenguaje muy popular gracias a sus principales paquetes orientados/usados en este ámbito. Algunos ejemplos son *Pandas*, *NumPy* y *SciPy*, que producen excelentes resultados para trabajos de análisis de datos. Si tenemos que emplear gráficos, *matplotlib* es una gran opción para ello, al igual que *scikit-learn* para tareas de *machine learning*.

---

Para el desarrollo del servicio de recomendación y la aplicación web, hemos utilizado diferentes lenguajes. Para implementar *API REST*, que se encargará de hacer las recomendaciones de playlists, hemos usando también *Python*. En cambio, para el desarrollo de la aplicación web, hemos utilizado *ASP.NET Core MVC* (empleando la plataforma *.NET 5.0*), con los lenguajes de programación *HTML* para el frontend y *C#* para el backend.

Por último, también hemos empleado *HCL (HashiCorp Configuration Language)* para desplegar la infraestructura en la nube. Dicho lenguaje se emplea para definir la infraestructura como código (IaC) mediante la herramienta *Terraform*, de la cual hablaremos más adelante.

### 3.2.2. Servicios en la nube

Para desplegar nuestra infraestructura, hemos elegido *Microsoft Azure* como proveedor de servicios en la nube. Aunque explicaremos con más detalle los servicios utilizados a lo largo de este documento conforme se vayan requiriendo, vamos a mencionarlos brevemente:

- **Azure Machine Learning Studio:** Área de trabajo que incorpora todos los elementos necesarios para realizar proyectos de *machine learning*, como por ejemplo: libretas Jupyter (incorporando un entorno propio para ejecutarlas), colecciones de modelos, experimentos, instancias de cómputo, etc.
- **Virtual Machines:** Uso de máquinas virtuales en la nube.
- **Azure Cosmos DB:** Base de datos NoSQL, de acceso rápido y con alta disponibilidad, respaldadas por un Acuerdo de Nivel de Servicio, escalabilidad automática e instantánea.
- **Azure Functions:** Servicio que proporciona toda la infraestructura y los recursos, que se actualizan continuamente, necesarios para ejecutar aplicaciones.
- **API Managment:** Espacio de trabajo donde podemos configurar distintas APIs (propias o de terceros) añadiendo una capa de abstracción que unifica el acceso de desarrolladores a ellas.
- **Azure Blob Storage:** Solución de almacenamiento de objetos en la nube.
- **Azure DevOps:** Servicio que proporciona control de versiones, informes, gestión de requisitos, gestión de proyectos, compilaciones automatizadas, pruebas y capacidades de gestión de versiones.

### 3.2.3. Herramientas de desarrollo

Para la creación del conjunto de datos, la creación del modelo, pruebas, análisis, etc., se han empleado los siguientes entornos:

- **Jupyter Notebook:** Entorno de programación interactivo, mediante el cual podemos introducir texto en formato *markdown* junto a código *Python* haciendo uso de los *notebooks* o libretas.
- **Visual Studio Code:** Conocido editor de código de Microsoft, el cual hemos empleado a la hora de visualizar los archivos que hemos creado para almacenar los datos, tanto en formato CSV como *JSON*, para el desarrollo de scripts empleados para ejecutar los procesos relacionados con la recopilación de datos y el entrenamiento del modelo, y para desarrollar la aplicación web y la *API REST*.

#### 3.2.4. Bibliotecas

Aunque hemos empleado numerosas bibliotecas o paquetes a la hora de realizar las tareas de desarrollo, vamos a nombrar algunas de las más destacadas:

- **Spotipy:** Biblioteca de Python para usar la *WebAPI* de *Spotify*.
- **Numpy:** Ofrece mayor soporte a vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar con estos elementos.
- **Pandas:** Es una biblioteca de código abierto para *Python* que proporciona estructuras de datos y herramientas de análisis de alto rendimiento y fáciles de emplear.
- **NLTK:** Este toolkit es una de las bibliotecas más potentes para procesamiento del lenguaje natural.
- **SciPy:** Biblioteca de código abierto basada en *Python*, que se utiliza en matemáticas, ciencia e ingeniería.
- **Scikit-Learn:** Biblioteca empleada para problemas de aprendizaje automático o *machine learning*.
- **LightFM:** Es una biblioteca que contiene numerosos algoritmos utilizados en los sistemas de recomendación.
- **Azure SDK:** Conjunto de herramientas para emplear los servicios creados en *Microsoft Azure*.
- **.NET 5:** Framework multiplataforma gratuito y *open source* que nos permite el desarrollo de aplicaciones. En particular utilizaremos *Blazor*, que viene integrado en el framework, y es un completo marco de trabajo para compilar aplicaciones web.

#### 3.2.5. Hardware empleado

Para las primeras fases de la creación del modelo, el desarrollo del servicio de recomendación (*API REST*) y la aplicación web, se ha empleado un ordenador personal, cuyas prestaciones podemos ver en la Tabla 3.1.

Equipo principal: Surface Book 2	
Procesador	Intel Core i7-8650U CPU @ 1.90GHZ, 2112 MHz, 4 procesadores principales, 8 procesadores lógicos.
Memoria RAM	16 GB
Almacenamiento	512 GB
Sistema operativo	<ul style="list-style-type: none"> <li>■ Windows 10 Pro (versión 21H1)</li> <li>■ Ubuntu 20.04 (bajo <i>Windows Subsystem for Linux</i>)</li> </ul>

**Tabla 3.1:** Especificaciones hardware del equipo principal

Para el entrenamiento del modelo, ha sido necesario iniciar una instancia de cómputo dentro de nuestra área de trabajo de aprendizaje automático en *Microsoft Azure*, cuyas prestaciones podemos ver en la Tabla 3.2 .

Instancia de cómputo	
Serie de VM	Serie Fsv2
Categoría	Optimizada para procesos
vCPU	32
Memoria RAM	64 GB
Almacenamiento	256 GB ( <i>Temporal</i> )
Procesador	Intel Xeon Platinum 8272CL (Cascade Lake) o Intel Xeon Platinum 8168 (Skylake)
Sistema operativo	Ubuntu 18.04 (personalizada con herramientas de <i>Machine Learning</i> )

**Tabla 3.2:** Especificaciones de la instancia de cómputo



## 4. Modelo de datos para la recomendación de playlists

---

A lo largo de este capítulo, describiremos el proceso realizado partiendo del conjunto de datos empleado hasta llegar a la creación de un punto de conexión (*endpoint*) para consumir los resultados que nos proporcionará el modelo de datos.

### 4.1. Infraestructura

Antes de comenzar, vamos a definir la infraestructura que necesitamos crear en la nube perteneciente a los procesos de aprendizaje automático. Esta infraestructura estará formada por instancias de servicios de *Machine Learning as a Service* (MLaaS) disponibles en Microsoft Azure. En particular, vamos a crear los siguientes recursos:

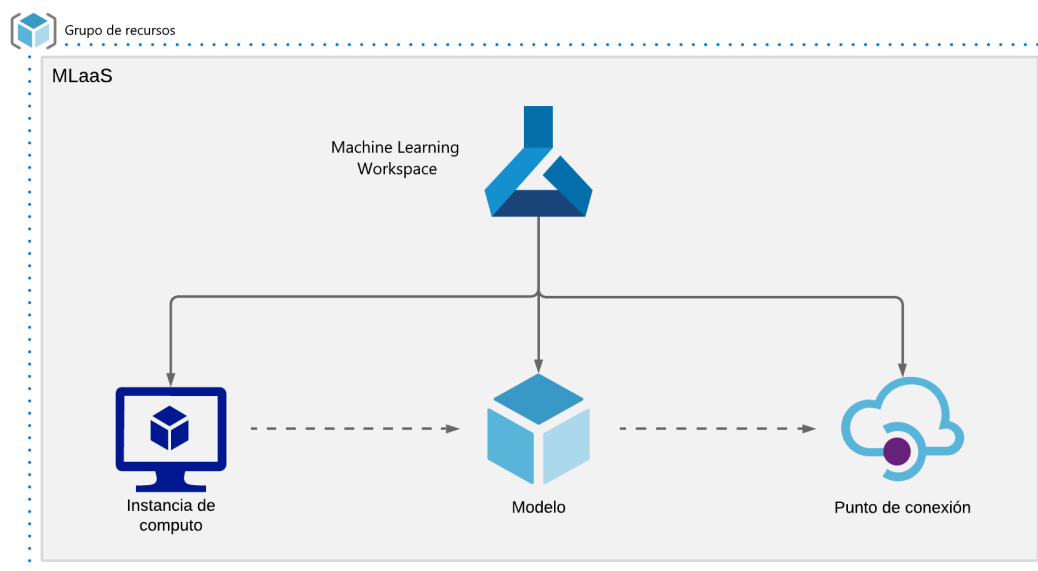
- Área de trabajo de *machine learning*.
- Instancia de proceso *Fsv2* para realizar el entrenamiento del modelo.
- Modelo.
- Punto de control o *endpoint* para consumir las predicciones realizadas por el modelo, una vez entrenado, haciendo uso de la información que falta en cada playlist.

Tanto la instancia de proceso *Fsv2*, como el modelo y el punto de conexión, serán creados dentro del área de trabajo de *machine learning* de forma programática. Adicionalmente, durante la creación del área de trabajo, también se crearán de forma automática los siguientes recursos:

- **Application Insights:** Recurso para obtener las métricas de los elementos que se encuentran bajo el área de trabajo de machine learning.

- **Key Vault:** Almacén de claves criptográficas y otras variables de carácter privado usadas por las aplicaciones y los servicios de *machine learning*.
- **Storage Account:** Cuenta de almacenamiento donde se guardarán los artefactos creados en el área de trabajo, tales como libretas, modelos, experimentos, etc ...

Todos estos recursos estarán agrupados en un grupo de recursos en *Microsoft Azure*. En la Figura 4.1 podemos ver un diagrama de los recursos que conforman la infraestructura para los procesos de aprendizaje automático.



**Figura 4.1:** Infraestructura para los procesos de aprendizaje automático en la nube

A modo de resumen, dentro del área de trabajo de *machine learning*, se creará una instancia de proceso para entrenar el modelo de recomendación de nuestro sistema. Una vez entrenado y obtenido el modelo, lo almacenamos en la colección de modelos y creamos un punto de conexión para consumir las predicciones de dicho modelo.

El despliegue de estos recursos se realizará mediante la herramienta *Terraform*, la cual explicaremos con más detalle en el Capítulo 6.

Podemos encontrar más detalles sobre la infraestructura en la libreta *Jupyter* titulada *Parte 4 - Infraestructura*.

## 4.2. Conjunto de datos

El conjunto de datos que hemos empleado en este proyecto es el obtenido en el Trabajo de Fin de Grado *Generación automática de playlist de canciones mediante técnicas de minería de datos* [11]. Dicho conjunto está basado en el *Million Playlist Dataset* de Spotify, empleado para la competición *RecSys Challenge* del año 2018 [33], y que ha sido liberado recientemente en *AlCrowd* para investigación [34].

Dicho conjunto de datos contiene 1.000.000 de playlists, creadas por los usuarios de la plataforma *Spotify*, e incluye datos como títulos de las listas de reproducción, pistas que pertenecen a cada una playlists, junto a sus correspondientes artistas y álbumes, número de seguidores, etc.

Adicionalmente, se proporciona otro conjunto de test, con información incompleta (sin título y con un determinado número de pistas, sólo título, ...), mediante el cual podemos evaluar el rendimiento de nuestro modelo.

## 4.3. Preprocesamiento de los datos

Lo primero que hemos hecho es convertir los datos de su formato original, *JSON*, a formato *CSV*. Este cambio lo realizamos para reducir su tamaño y poder almacenarlo en memoria, además de ser un formato compatible con librerías como *numpy* o *pandas*.

También hemos realizado una serie de modificaciones adicionales para eliminar información innecesaria, o repetida, como los prefijos de los identificadores de pistas, artistas o álbumes. Para ello, también exportaremos información a otros ficheros *CSV* para tenerla mejor clasificada.

Una vez concluido el preprocesamiento de los datos, hemos obtenido varios conjuntos (ficheros *CSV*). Estos conjuntos son los siguientes:

- Conjunto con la información sobre las playlists (nombre, número de pistas, duración, seguidores, etc.).
- Conjunto con la información sobre las playlists del conjunto de prueba.
- Conjunto con la información sobre los álbumes a los que pertenecen las pistas.
- Conjunto con la información relativa a los artistas a los que pertenecen las pistas de las playlists.
- Conjunto que contiene las pistas de una playlist junto a su posición.

Para futuros desarrollos o investigaciones, hemos creado dos conjuntos extra:

- 
- Conjunto que contiene los artistas que aparecen en una playlist junto al número de apariciones.
  - Conjunto que contiene los álbumes que aparecen en una playlist junto al número de apariciones.

Podemos ver todo el preprocesamiento de los datos en la libreta *Jupyter* titulada *Parte 1 - Preparación de datos*.

## 4.4. Obtención de datos adicionales

Con el fin de realizar mejores predicciones, hemos obtenido información adicional sobre las pistas que conforman una playlist empleando *Spotipy* para acceder a la *API* de *Spotify*.

Mediante uso de técnicas de Procesamiento del Lenguaje Natural o *NLP* (*Natural Language Processing*), también hemos procesado los títulos de las playlists para convertirlos en etiquetas o *tokens*, con lo cual conseguiremos relacionar playlists de títulos similares.

En la libreta *Jupyter* titulada *Parte 2 - Obtención de datos adicionales* podemos encontrar el proceso completo.

### 4.4.1. Creación de etiquetas

Para la creación de etiquetas, con las que identificamos las playlists y que nos ayudarán a relacionarlas con otras similares, partimos de sus títulos y los hemos procesado de manera que queden en un formato común (todos los caracteres en minúscula, sin signos de puntuación, ...). También identificamos aquellas playlists cuyo nombre contiene algún emoticono y los transformamos a texto, todo ello obteniendo los términos más frecuentes que aparecen junto a cada emoticono.

Para la creación del diccionario de emoticonos, hemos realizado los siguientes pasos:

1. Extracción de los emoticonos que posee una lista.
2. Transformación del título de la lista al formato común que hemos establecido.
3. Búsqueda de las playlists en las que aparece cada emoticono y extracción de las palabras que le acompañan con mayor frecuencia.

En caso de que un emoticono no aparezca acompañado de ninguna palabra, lo dejaremos como tal en el texto normalizado. Una vez concluido el proceso, obtenemos las posibles traducciones a texto de un emoticono, tal como mostramos en el ejemplo de la Figura 4.2.

{ 🍁 :	fall	}
{ ☀️ :	summer	}
{ 🎸 :	country, rock	}
{ 🎅 :	christmas	}
{ 🎃 :	halloween	}
{ 💪 :	workout, gym	}

**Figura 4.2:** Muestra del diccionario de traducción de emoticonos

Una vez que hemos obtenido el diccionario, hemos procedido a crear las etiquetas que se corresponderán con las playlists. Para ello, hemos vuelto a aplicar el mismo método que empleamos anteriormente para normalizar los títulos, con la diferencia de que esta vez añadimos el paso de traducción de emoticonos y devolvemos una lista con las etiquetas.

### Construcción de la matriz TF-IDF

Para poder entrenar nuestro modelo de recomendación a partir de un conjunto de documentos, en nuestro caso los títulos de las playlists, es necesario representar estos datos en una matriz bidimensional. En el modelo *Bag of Words*, cada documento se representa a partir del conjunto de palabras que aparecen en él. Como paso previo a la construcción de la matriz de datos, se elabora un vocabulario con la unión de todos los términos que aparecen en algún documento. A partir de éste, se construye una matriz en la que cada fila representa un documento, y cada columna (cada característica) corresponde a un término [35].

En la versión más básica, cada posición  $(d, t)$  de la matriz contiene el valor para la medida *Term Frequency* (frecuencia de términos), que se denota  $tf_{d,t}$ , y refleja al número de veces que aparece el término  $t$  en el documento  $d$  [35]. Por ejemplo, dados estos tres documentos:

- Documento 1: *El objetivo de esta práctica es explicar el procesamiento básico de texto libre,*
- Documento 2: *Uno de los enfoques utilizados es la bolsa de palabras,*
- Documento 3: *Procesamiento de texto mediante bolsa de palabras*

la medida *Term Frequency* generaría esta matriz de datos:

$$X = \begin{bmatrix} El & objetivo & de & esta & \cdots & bolsa & palabras \\ 2 & 1 & 2 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 2 & 0 & \cdots & 1 & 1 \\ 0 & 0 & 2 & 0 & \cdots & 1 & 1 \end{bmatrix} \begin{matrix} \text{Documento 1} \\ \text{Documento 2} \\ \text{Documento 3} \end{matrix}$$

---

En la matriz,  $tf_{0,1} = 2$ , porque la palabra de índice 0, *El*, aparece dos veces en el Documento 1.

Existen algunas variantes de esta medida, de modo que la matriz de datos se puede formar con:

- $tf_{d,t} \in \{0, 1\}$  - Es decir, si el término aparece o no en el documento.
- $\log(1 + tf_{d,t})$  - Escala logarítmica.
- $tf_{d,t} / \max_j tf_{d,j}$  - Escala en relación al término más frecuente en el documento.

La frecuencia de términos comunes como *el*, *a*, *de*, etc, suele ser alta para la mayoría de los documentos. La métrica *Inverse Document Frequency* (frecuencia inversa en documentos) penaliza los términos en proporción a la frecuencia con que aparecen en el conjunto de documentos [35]. La frecuencia inversa de un término se formula como:

$$idf_t = \log \left( \frac{\# \text{ documentos}}{\# \text{ documentos que incluyen la palabra } t} \right)$$

En el ejemplo anterior:

- $idf_{el} = \log \left( \frac{3}{1} \right) = 1.098$
- $idf_{objetivo} = \log \left( \frac{3}{1} \right) = 1.098$
- $idf_{de} = \log \left( \frac{3}{3} \right) = 0$
- $idf_{esta} = \log \left( \frac{3}{1} \right) = 1.098$
- $idf_{bolsa} = \log \left( \frac{3}{2} \right) = 0.405$
- $idf_{palabras} = \log \left( \frac{3}{2} \right) = 0.405$

Existen algunas formulaciones alternativas como:

$$idf_t = \log \left( \frac{1 + \# \text{ documentos}}{1 + \# \text{ documentos que incluyen la palabra } t} \right) + 1$$

Por último, lo habitual es utilizar una combinación de estas métricas, denominada *Term Frequency Inverse Document Frequency* (*tf-idf*), que se formula como:

$$tf - idf_{d,t} = tf_{d,t} \times idf_t$$

Dados los documentos anteriores, la matriz con los valores  $tf - idf_{d,t}$  sería:

$$X = \begin{bmatrix} El & objetivo & de & esta & \dots & bolsa & palabras \\ 2.2 & 1.1 & 0 & 1.1 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0.41 & 0.41 \\ 0 & 0 & 0 & 0 & \dots & 0.41 & 0.41 \end{bmatrix} \begin{matrix} Documento 1 \\ Documento 2 \\ Documento 3 \end{matrix}$$

Una vez que ha sido creada la matriz *tf-idf*, empleando la función que hemos definido para *tokenizar* el texto de la forma antes mencionada, creamos un diccionario con dos claves:

- **features\_list**, cuyo valor es el diccionario con las características obtenidas.
- **matrix**, cuyo valor es la matriz *tf-idf*.

y lo almacenamos en un fichero *pickle*.

#### 4.4.2. Obtención del género musical de los artistas

El primer dato adicional que vamos a obtener, son los géneros musicales de los artistas que conforman nuestro conjunto de datos (puesto que *Spotify* no proporciona los géneros para pistas y/o álbumes). Usaremos esta información para etiquetar las pistas que pertenezcan a un artista con los mismos géneros musicales. Antes de guardar la información en un fichero CSV, concatenamos todos los géneros en un único *string* empleando el carácter `'|'` como separador.

#### 4.4.3. Obtención de las fechas de lanzamiento de los álbumes

Habitualmente, se suelen crear playlists que contienen pistas que pertenecen a un año, década o periodo de tiempo determinado, recopilando lo mejor de ese tiempo, canciones favoritas de una época, etc. Como *Spotify* proporciona la fecha de lanzamiento de los álbumes que posee, vamos a obtener dicha información para el conjunto de álbumes que componen nuestro conjunto de datos. Una vez obtenidas las fechas, las almacenamos en un nuevo fichero CSV.

### 4.5. Selección del modelo: LightFM

*LightFM* es una implementación en *Python* de una serie de algoritmos de recomendación para la retroalimentación implícita y explícita, incluida la implementación eficiente de las funciones de pérdida *BPR* y *WARP*. Es fácil de usar, rápido (a través de la estimación de modelos multiproceso) y produce resultados de alta calidad.

---

También permite incorporar metadatos de elementos y usuarios en los algoritmos tradicionales de factorización matricial, representando a cada usuario y elemento como la unión de las representaciones latentes de sus características, permitiendo así que las recomendaciones se generalicen a nuevos elementos (a través de las características del elemento) y a nuevos usuarios (a través de las características del usuario).

*LightFM* es un modelo híbrido de recomendación. Dicho modelo aprende representaciones latentes en un espacio de alta dimensión para usuarios y elementos de manera que codifica las preferencias del usuario sobre los elementos [36].

Las representaciones de usuario y elemento son expresadas en función de sus características. Por ejemplo, si la película *El caballero oscuro* se describe mediante las siguientes características: *acción*, *superhéroes* y *comics*, su representación estará formada por la unión de las representaciones latentes de estas características. Las incrustaciones se aprenden a través de métodos de descenso estocástico del gradiente [36].

A continuación, se muestran algunas de las funciones de pérdida que están presentes en *LightFM*:

- **BPR (Bayesian Personalised Ranking)**: Maximiza la diferencia de predicción entre un ejemplo positivo y un ejemplo negativo elegido al azar. Es útil cuando sólo hay interacciones positivas y se desea optimizar el área bajo la curva ROC (AUC) [36].
- **WARP (Weighted Approximate-Rank Pairwise)**: Maximiza el rango de ejemplos positivos al muestrear repetidamente ejemplos negativos hasta que se encuentra el rango que lo incumpla. Es útil cuando sólo hay interacciones positivas y se desea optimizar la parte superior de la lista de recomendaciones (*precision@k*) [36].

En *LightFM* también nos encontramos con dos algoritmos de optimización para gradiente descendiente: *Adagrad* y *Adadelta* [36].

Todas las tareas que se detallan en esta sección se encuentran en la libreta *Jupyter* titulada *Parte 3 - Modelo de recomendación*.

## Motivos de la elección de LightFM

La principal razón por la cual hemos optado por la librería *LightFM* es el uso de un sistema de recomendación híbrido, que resulta de combinar el filtrado colaborativo y el filtrado basado en contenido.

Otra de las razones por las cuales hemos optado por ella, es el problema del arranque en frío. En nuestro proyecto, debemos ser capaces de realizar una recomendación basada en las características del usuario y sin tener ninguna valoración previa de playlists o canciones por parte de éste [37].



```
class lightfm.LightFM(no_components=10, k=5, n=10,
                      learning_schedule='adagrad', loss='logistic',
                      learning_rate=0.05, rho=0.95, epsilon=1e-06,
                      item_alpha=0.0, user_alpha=0.0,
                      max_sampled=10, random_state=None)
```

---

**Código 4.1:** Definición del modelo LightFM

---

#### 4.5.1. Conversión del conjunto al formato admitido

Aunque ya hemos realizado el proceso de preprocesamiento de datos, necesitamos adaptar los datos al formato admitido por *LightFM*. Para ello, necesitamos crear las siguientes matrices:

- **Matriz de interacción:** Contiene las interacciones entre los usuarios (filas) y los ítems (columnas), cuyos posibles valores son 1 ó 0 según haya o no interacción.
- **Matriz de características de usuarios:** Indica las características (columnas) que posee un usuario (filas).
- **Matriz de características de ítems:** Indica las características (columnas) que posee un ítem (filas).

Para ello, emplearemos la librería '*Data*' que proporciona *LightFM* junto al módulo de matrices de expansión de *SciPy*.

#### 4.5.2. Definición y entrenamiento del modelo

A continuación, vamos a describir cómo hemos definido el modelo de *LightFM* que hemos empleado en este trabajo. En el Código 4.1 podemos ver qué parámetros recibe la función que se encarga de crear el modelo, junto a los valores que toma por defecto.

La decisión que hemos tomado para establecer los parámetros del modelo, ha sido emplear como referencia algunos de los valores que aparecen en el artículo *Learning to Rank Sketchfab Models with LightFM* [38]. Dichos parámetros son los siguientes:

- **Función de pérdida (loss):** WARP (*Weighted Approximate-Rank Pairwise*). Hemos elegido esta función ya que únicamente tenemos interacciones positivas y pretendemos optimizar la parte superior de la lista de recomendaciones.
- **Número máximo de muestras (max\_sampled):** 30. Hemos elevado el número de muestras para mejorar la precisión del modelo.

- 
- **Número de componentes (`no_components`):** 200. Se ha elegido este valor de entre los propuestos [38] para que cada ítem tenga un número de características amplio con las que ser definido, procurando que el tiempo de entrenamiento no sea muy elevado.
  - **Penalización para las características (`user_alpha`):**  $10^{-6}$ . Se minimiza la penalización recomendada.

Para los parámetros restantes, *learning\_schedule* y *learning\_rate*, hemos decidido dejar los valores por defecto.

Una vez que hemos establecido las características del modelo *LightFM*, iniciamos la instancia de cómputo que hemos creado para el entrenamiento y, mediante el uso de un *script* de *Python*, ejecutamos el proceso de entrenamiento. Dicho *script* se encarga de leer el fichero con los datos de entrenamiento del modelo (matrices de interacción, características de playlists y características de pistas), pasárselos al modelo y realizar el entrenamiento. Concluido el entrenamiento, se almacena el modelo en un fichero *pickle* y lo registra en el área de trabajo de *machine learning*, que habíamos creado previamente, para poder emplearlo en el punto de conexión. Finalizado el proceso, se apaga la máquina empleada para el entrenamiento.

El tiempo empleado para el entrenamiento del modelo, con la instancia de cómputo cuyas características se encuentran en la Tabla 3.2, estableciendo 75 ciclos (o *epochs*), ha sido de 4 horas (aproximadamente).

## 4.6. Creación del punto de conexión al modelo

Con el modelo ya entrenado, únicamente nos queda montar el punto de conexión que se encargara de ofrecer los resultados ofrecidos por dicho modelo. Este proceso es muy sencillo y lo realizaremos de forma programática.

Todo el proceso completo para la creación de este *endpoint* podemos encontrarlo en la libreta *Jupyter* titulada *Parte 5 - Servicios de Machine Learning*.

Desde dicha libreta hemos procedido a acceder a nuestra área de trabajo de aprendizaje automático en *Microsoft Azure* para poder acceder al modelo entrenado y asociarle el punto de conexión que vamos a crear.

Para el punto de conexión, necesitamos crear un *script* de entrada que se encarga de recoger la petición que se le envía con los datos necesarios, interpretarlos, realizar la recomendación empleando el modelo seleccionado y devolver el resultado en el formato adecuado. Este *script* se caracteriza por tener dos funciones principales:

- **`init`:** Se encarga de leer el modelo y sus dependencias almacenadas en formato *pickle* de nuestra colección de modelos del área de trabajo.

- **run**: Se encarga de transformar los datos al formato admitido por el modelo, realizar la predicción y devolver los resultados obtenidos.

En nuestro caso, el modelo es capaz de ofrecernos varios tipos de predicciones que devuelven distintos tipos de información. Las funciones que se encargan de realizar estas tareas son:

- **similar\_users**: Se encarga de buscar playlists similares a la indicada.
- **similar\_items**: Se encarga de buscar pistas similares a la indicada.
- **get\_similar\_user\_tags**: Obtiene las características de las playlists que están relacionadas con la que se le indica.
- **get\_similar\_item\_tags**: Obtiene las características de las pistas que están relacionadas con la que se le indica.
- **make\_newuser\_recomm**: Realiza una recomendación de pistas sobre una playlist que no se encuentra en nuestro conjunto, únicamente empleando el título de esta (problema del *cold-start*).
- **make\_user\_recomm**: Realiza una recomendación de pistas sobre una playlist que se encuentra en nuestro conjunto de datos.

Una vez definido el *script de entrada*, pasamos a configurar las características que tendrá el contenedor con el modelo y el *script* que realizará las predicciones. A este contenedor se le conoce con el nombre de *Azure Container Instance* en nuestro proveedor de servicios en la nube. Las características hardware que hemos establecido para este contenedor son las siguientes

- **vCPU**: 3 núcleos
- **Memoria RAM**: 12 GB

A continuación pasamos a definir el *environment* de *Python* (empleando *Conda*), sobre el que se ejecutará nuestro *script* con las dependencias necesarias. Posteriormente, también definimos la configuración de inferencia (que es la configuración que contiene el entorno *Python* a emplear y la configuración del *script de entrada*) que le indicaremos al punto de conexión.

Con toda la configuración necesaria ya realizada, pasamos a crear el punto de conexión (*endpoint*) y a comprobar que se reciben correctamente las solicitudes realizadas.



## 5. Servicio de recomendación de playlists

---

Una vez que ya tenemos disponible el punto de acceso con el modelo encargado de realizar las predicciones, vamos a montar el servicio web para la recomendación de playlists y la aplicación web que va a hacer uso de él.

Durante este capítulo, veremos la arquitectura empleada para montar el servicio, junto a las tecnologías y recursos que hemos utilizado.

### 5.1. Informática sin servidor

Antes de empezar a ver la infraestructura, servicios y tecnología que hemos empleado, vamos a hacer una breve introducción al concepto *informática sin servidor* para poder comprender los servicios usados para implementar el servicio de recomendación.

La informática sin servidor, o *serverless computing*, permite a los desarrolladores crear aplicaciones de forma rápida, ya que no es necesario que administren la infraestructura. Con las aplicaciones sin servidor, el proveedor de servicios en la nube aprovisiona, escala y administra automáticamente la infraestructura necesaria para ejecutar el código [39].

Para entender la definición de la informática sin servidor, es importante tener en cuenta que los servidores siguen ejecutando el código. El término sin servidor significa que las tareas asociadas con el aprovisionamiento y la administración de la infraestructura son invisibles para el desarrollador. Este enfoque permite a los desarrolladores centrarse más en la lógica de negocios y en aportar más valor al núcleo principal del negocio. La informática sin servidor ayuda a los equipos a aumentar su productividad y a comercializar los productos más rápido, además de permitir a las organizaciones optimizar mejor los recursos y seguir centrándose en la innovación [39].

Las principales ventajas que nos ofrece este modelo de servicios en la nube, son las

---

siguientes [39]:

- **No es necesaria la administración de la infraestructura:** El uso de servicios totalmente administrados, permite a los desarrolladores evitar las tareas administrativas y centrarse en la lógica de negocios principal.
- **Escalabilidad dinámica:** Con la informática sin servidor, la infraestructura se escala y reduce verticalmente de forma dinámica en cuestión de segundos, a fin de satisfacer la demanda de cualquier carga de trabajo.
- **Comercialización más rápida:** Las aplicaciones sin servidor reducen las dependencias de las operaciones de cada ciclo de desarrollo, con el aumento de la agilidad de los equipos de desarrollo para ofrecer más funcionalidad en menos tiempo.
- **Uso eficaz de los recursos:** La migración a las tecnologías sin servidor ayuda a las organizaciones a reducir el coste total de propiedad y a reasignar los recursos para acelerar el ritmo de la innovación.

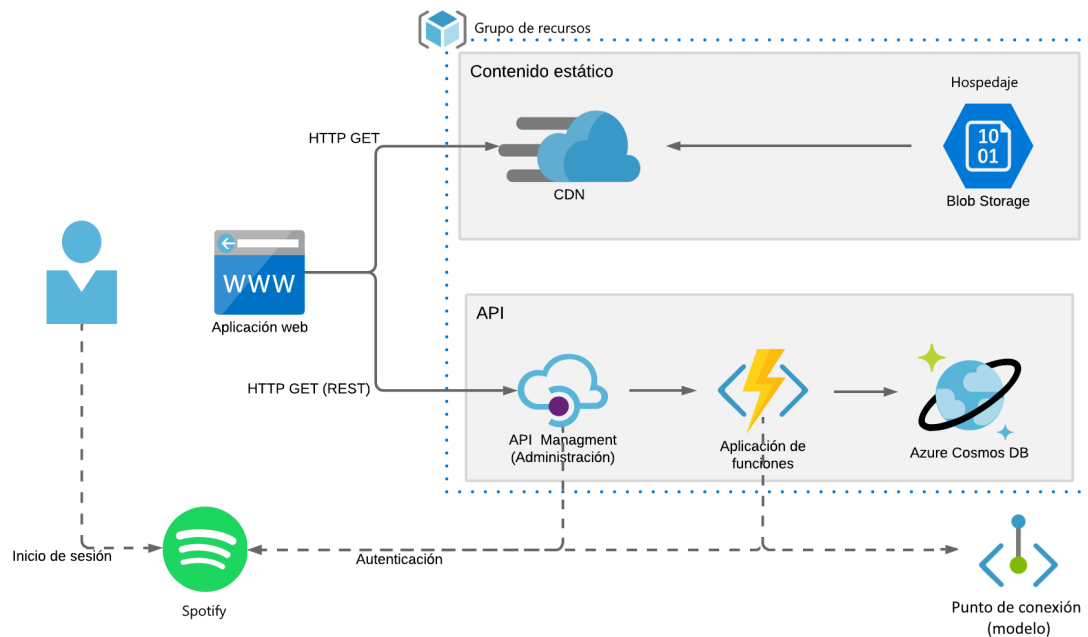
## 5.2. Infraestructura

La infraestructura de la cual se compone nuestro servicio de recomendación de playlists, hace uso de los siguientes servicios *serverless*:

- Base de datos (*Azure Cosmos DB*).
- Aplicación de funciones (*Azure Functions*).
- Servicio *API Rest* (*API Management*).
- Aplicación web alojada en un *blob* (contenedor de archivos) dentro de nuestra cuenta de almacenamiento.

Dichos servicios se irán explicado a lo largo de las distintas secciones que componen este capítulo. En la Figura 5.1 podemos ver el diseño de la infraestructura del servicio de recomendación y cómo están conectados sus distintos componentes.

En la base de datos que hemos creado en *Azure Cosmos DB*, se encuentra almacenada toda la información relativa a las pistas, artistas, álbumes, playlists, etc. Esta información es accesible mediante *Azure Functions*, que también se encarga de conectarse al punto de conexión del modelo de recomendación de playlists y de realizar la autenticación en *Spotify* para leer las playlists de un usuario. Mediante *API Management* enlazamos las funciones con el servicio *API REST* que hemos definido, al igual que se atienden las peticiones de autenticación de *Spotify*. Por último, la aplicación web que hemos creado se encuentra alojada en un contenedor de archivos (*blob storage*) que también hace uso de la *API REST*.



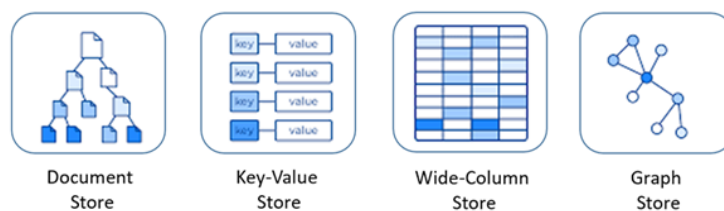
**Figura 5.1:** Infraestructura del servicio de recomendación de playlists

### 5.3. Base de datos

Para alojar nuestra base de datos hemos empleado *Azure Cosmos DB*, un servicio de base de datos distribuido con escalado horizontal, que nos permite distribuir datos de forma global sobre cualquier número de regiones de *Microsoft Azure*, empleando un proceso transparente de escalado y replicación de los datos. También garantiza valores de latencia inferiores a 10 milisegundos en cualquier parte del mundo. Además, ofrece varios modelos de coherencia bien definidos para ajustar el rendimiento y garantizar alta disponibilidad con hospedaje múltiple [40] [41].

A nivel de funcionalidad, *Azure Cosmos DB* indexa los datos automáticamente sin que haya que ocuparse de la administración de esquemas ni de índices. También se caracteriza por ser multimodelo, como podemos ver en la Figura 5.2, admitiendo de forma nativa los modelos que se detallan a continuación: [41] [42]:

- **SQL API:** Documentos *JSON* y consultas basadas en *SQL*.
- **Azure Table Storage:** Datos en formato clave-valor.
- **MongoDB API:** Documentos *JSON* y uso de la *API* de *MongoDB*.
- **Gremlin API:** Soporta la *API* de *Gremlin*.
- **Cassandra DB:** Para representaciones de datos en columnas.



**Figura 5.2:** Modelos admitidos por Cosmos DB [42]

En nuestro caso hemos empleado la API de *MongoDB*, que emplea documentos de tipo *JSON*, con la librería *PyMongo* que utiliza *Python* para trabajar con *MongoDB*. Nuestra base de datos, llamada *music*, está compuesta por las siguientes colecciones de documentos:

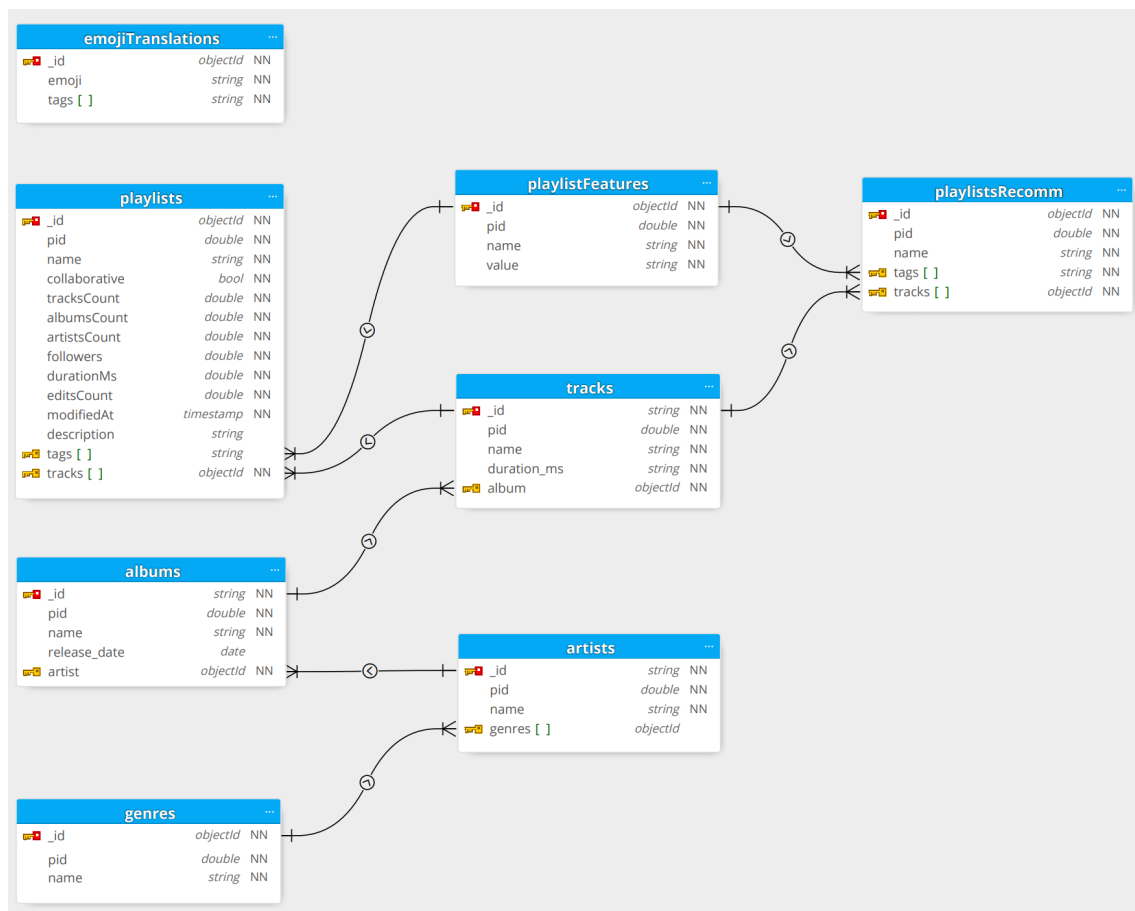
- *albums*: Contiene la información básica de los álbumes a los que pertenecen las pistas de las playlists.
- *artists*: Contiene todos los intérpretes a los cuales pertenecen los álbumes, así como a los géneros a los que pertenecen.
- *genres*: Contiene todos los géneros en los cuales se pueden clasificar los artistas.
- *playlists*: Contiene toda la información relativa a las listas de canciones, tales como el número de seguidores, si es colaborativa, última modificación, etc.
- *playlistsRecomm*: Esta colección la emplearemos para almacenar, de forma temporal, las playlists que recomiende nuestro sistema cuando solamente se le indique el título y/o características. Hemos creado esta colección para reducir el tiempo de espera y la carga al punto de conexión del modelo.
- *playlistFeatures*: Contiene las características/etiquetas con las que se pueden asociar las playlists.
- *tracks*: Colección con todas las pistas que existen en las playlists del sistema.
- *emojiTranslations*: Contiene el conjunto de emoticonos que pueden ser traducidos a texto, empleado para normalizar los títulos de las playlists y crear sus correspondientes etiquetas (playlist features).

En la Figura 5.3 se muestra la arquitectura de la base de datos que he hemos diseñado para nuestro proyecto.

## 5.4. Aplicación de funciones

Una aplicación de funciones de *Azure Functions*, es una solución del tipo *función como servicio* o *FaaS* (*Function as a Service*), que nos permite ejecutar fácilmente pequeñas piezas



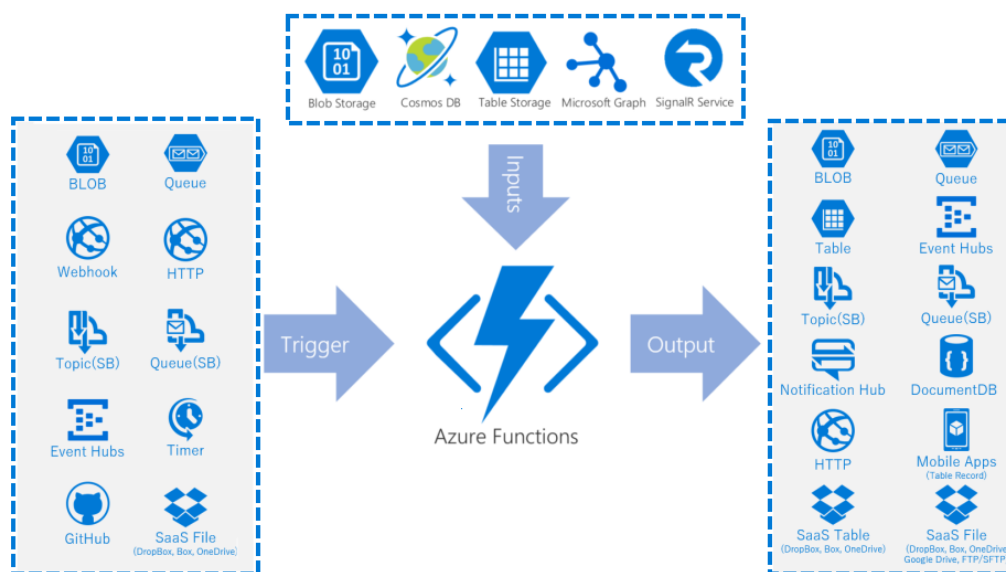


**Figura 5.3:** Arquitectura de la base de datos del sistema de recomendación

de código, conocidas como *funciones*, en la nube. Podemos escribir únicamente el código necesario para realizar la funcionalidad que deseemos, sin tener que preocuparnos de una aplicación completa o de la infraestructura para ejecutarla. Estas funciones utilizan distintos lenguajes de desarrollo, como C#, Python, Java o JavaScript, entre otros [43] [44].

El coste que se nos factura es el tiempo de ejecución del código de nuestra función. Además, si tenemos muchas peticiones a nuestra función, Microsoft Azure escalará de forma elástica a más instancias o menos en función de la cantidad de peticiones que nuestra función tenga que atender [45] [43].

En la Figura 5.4 podemos ver el esquema de funcionamiento de las funciones. Un desencadenador, o *trigger*, hace que la función se ejecute con la entrada que se le proporciona, ya esté indicada en el *trigger* o enlazada en el código de dicha función. A continuación, se ejecutará con los datos proporcionados y producirá una salida, siendo del tipo admitido por el sistema (respuesta HTTP, fichero a almacenar en un *blob*, documento a insertar en una base de datos, etc.).



**Figura 5.4:** Esquema de funcionamiento de una Azure Function

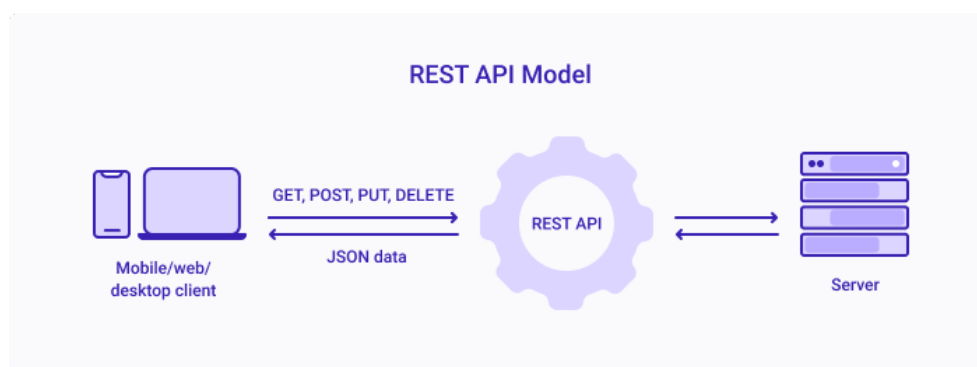
Para nuestro proyecto, se han definido una serie de funciones con tareas específicas para la recomendación de playlists, búsqueda de información en la base de datos, o interacción con *Spotify* (como la obtención de las playlists del usuario o la incorporación de las recomendadas por nuestro sistema). A continuación se detallan las funciones desarrolladas:

- **GetEquivalentPlaylist:** Busca una playlist, a partir de otra, que sea igual o similar. Consideramos que es igual si contiene las mismas canciones y el título, convertido a etiquetas, es el mismo. En caso de no encontrar ninguna igual, busca que las etiquetas creadas con el título y/o las canciones que contiene sean un subconjunto de otra playlist.
- **GetPlaylistRecs:** Mediante el identificador que tiene una playlist en la base de datos, devuelve un conjunto de playlist que recomienda el sistema.
- **GetPlaylistNewRecs:** A partir de un título, lo normaliza para convertirlo en etiquetas y llama al *endpoint* del modelo de recomendación para obtener una lista de canciones. Una vez obtenidas las canciones, se seleccionan algunas de forma aleatoria y se crea una playlist con el resultado.
- **GetPlaylistInfo:** Función que se encarga de completar una playlist, cuya lista de pistas es un conjunto de identificadores, añadiendo información de la pista, del álbum y del artista al que pertenece.
- **GetPlaylistInfo:** Igual que la función anterior, pero se emplea en el caso de obtener varias playlists sobre las que se requiere completar la información.

- **PostNewPlaylist:** Mediante una playlist, identificador de usuario y código de autorización temporal, crea dicha playlist en la cuenta de *Spotify* del usuario indicado.
- **GetUserPlaylist:** Obtiene la playlist indicada de la colección del usuario, ya sea pública o privada.
- **GetUserPlaylists:** Igual que el caso anterior, pero para obtener varias playlists.

## 5.5. API REST

A modo de recordatorio, una *API REST* es la implementación de una interfaz de programación de aplicaciones (*API*) que se adhiere a las restricciones de la arquitectura *REST*, actuando como interfaz. La comunicación entre el cliente y el servidor se realiza a través de las metodologías *HTTP*. En la Figura 5.5 podemos ver un esquema de funcionamiento [46].

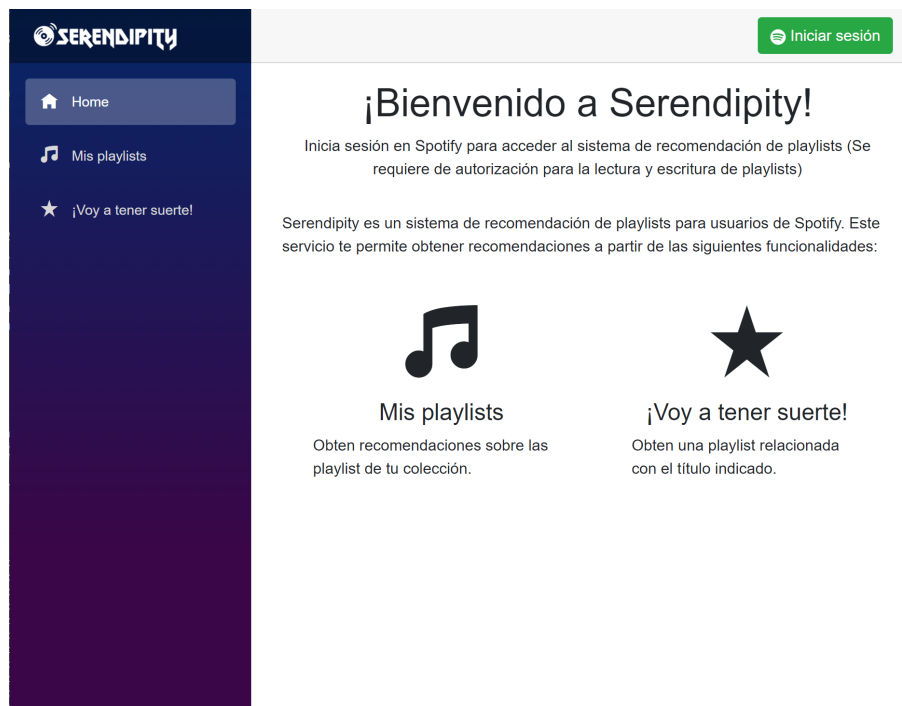


**Figura 5.5:** Esquema de funcionamiento de una API REST

En nuestro caso, agruparemos las funciones que hemos definido en nuestro proyecto de *Azure Functions* como una *API REST* mediante *API Managment*, un espacio de trabajo que nos proporciona *Azure* en el que, básicamente, podemos configurar distintas *APIs* (propias o de terceros) añadiendo una capa de abstracción que unifica el acceso de desarrolladores a ellas [47].

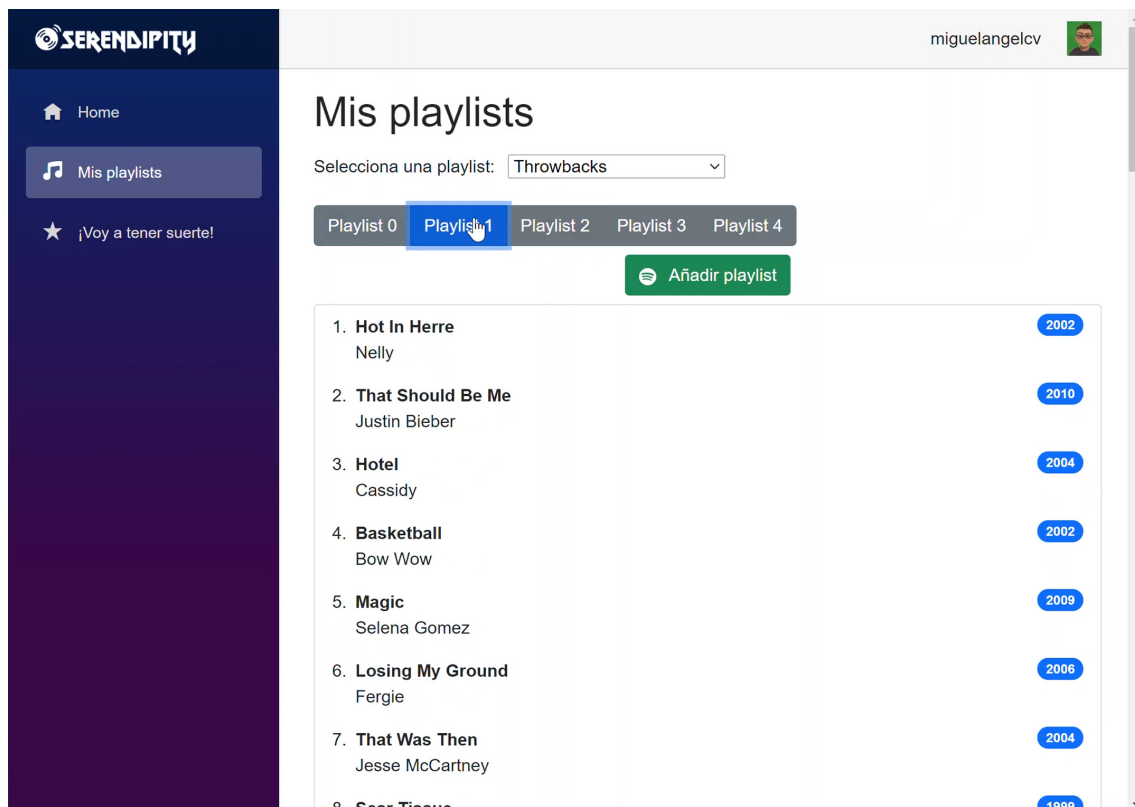
*Azure API Management* admite la importación de instancias de *Azure Functions* como nuevas *APIs* o anexionándolas a un proyecto existente. El proceso genera automáticamente una clave de host en la aplicación de funciones y le asigna un valor con nombre en *API Management*.

Los *endpoints* que se han obtenido tras crear la *API REST* empleando el proyecto de *Azure Functions* en *API Managment*, son los siguientes:



**Figura 5.6:** Página principal de la aplicación web

- **/recomm** : Realiza recomendaciones para playlists según su *id* (está en el sistema) o según su lista de canciones y/o título.
  - **GET**        /recomm/playlists/{id}
  - **GET**        /recomm/playlists
- **/playlists** : Se encarga de obtener playlists que se encuentran en nuestro sistema según su *id*, título, lista de canciones, etc. También es capaz de localizar una playlist lo más similar posible a una dada, para posteriormente realizar predicciones o buscar recomendaciones.
  - **GET**        /playlists
  - **GET**        /playlists/{id}
  - **GET**        /playlists/equivalent
- **/spotify** : Empleado para obtener playlists del usuario en *Spotify* y para añadir las playlists que ha recomendado nuestro sistema en caso de que el usuario lo indique.
  - **GET**        /spotify/playlists
  - **GET**        /spotify/playlists/{id}
  - **POST**       /spotify/playlists



**Figura 5.7:** Opción "Mis playlists" de la aplicación web

## 5.6. Aplicación web

Para el desarrollo de la aplicación web, que empleara el servicio de recomendación que hemos creado, vamos a usar *Blazor*.

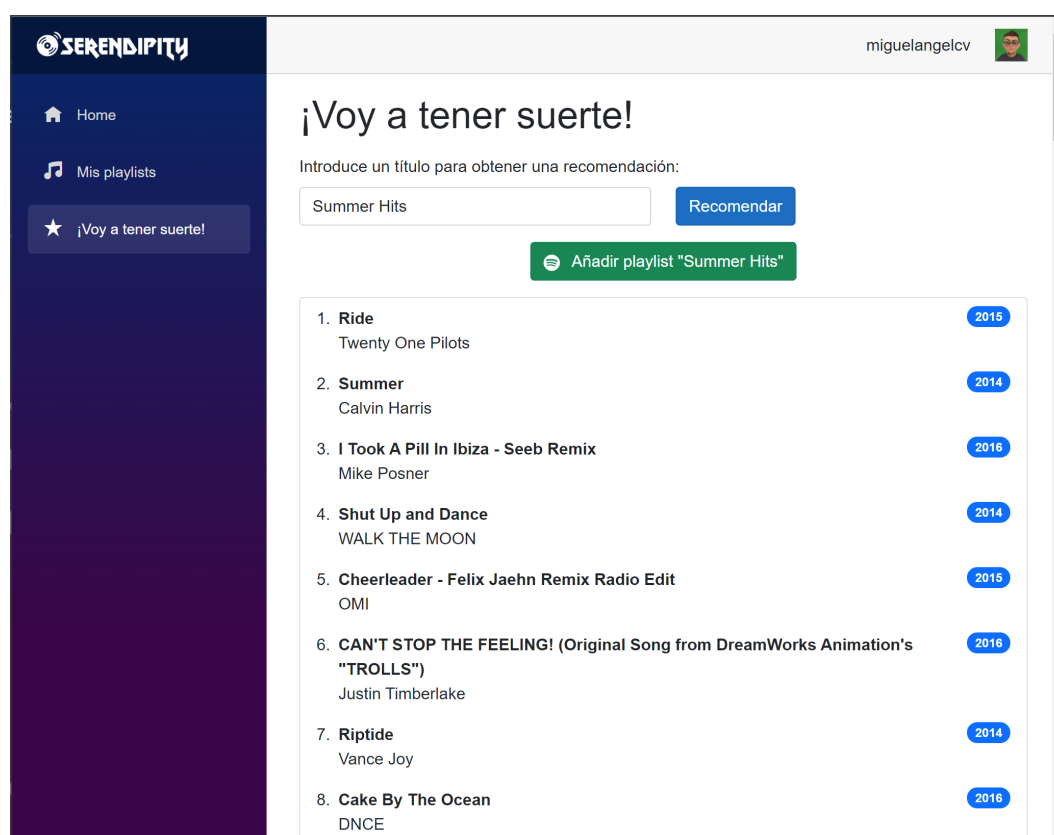
*Blazor* es un proyecto desarrollado por *Microsoft* creado para permitir crear SPAs (*Single Page Applications*) únicamente usando como lenguajes de programación *C#* y *Razor Pages*, haciendo nula la necesidad de programar en *Javascript* o frameworks derivados.

La aplicación web que hemos desarrollado, la cual hace uso del servicio de recomendación que hemos creado, nos permite ver algunas de las funciones que se pueden implementar mediante el uso del modelo de recomendación.

En la pantalla inicial de la aplicación, como se muestra en la Figura 5.6, avisamos al usuario del servicio que debe iniciar sesión en su cuenta de *Spotify* y autorizarla para leer sus playlists y crear nuevas listas.

Nuestra aplicación web realizara recomendaciones de playlists para usuarios de *Spotify* mediante dos formas:

1. **Mis playlists:** En esta primera forma para la recomendación de playlists, accedemos a la cuenta *Spotify* del usuario y obtenemos las playlists de su colección. Una vez obtenidas, el usuario selecciona aquella sobre la cual quiere obtener recomendaciones y se le ofrecen varias playlists, pudiendo elegir cualquiera de ellas y almacenarla en su cuenta. En la Figura 5.7 se muestra un ejemplo de esta opción.
2. **¡Voy a tener suerte!:** Ofrecemos al usuario la posibilidad de crear una playlist únicamente indicando un título. De esta forma podemos ver el redimiendo de nuestro modelo en aquellos casos de los que no se dispone de pistas (caso del *arranque en frío*). En la Figura 5.8 se muestra el resultado de la consulta *Summer Hits*.



**Figura 5.8:** Captura de pantalla de la opción "¡Voy a tener suerte!"

## 6. DevOps

---

Uno de los objetivos de este Trabajo de Fin de Máster es el uso de la metodología *DevOps* para mejorar el desarrollo de nuestro proyecto en menos tiempo y tener la capacidad de publicar más rápidamente revisiones o implementar nuevas funcionalidades. A lo largo de este capítulo haremos una breve introducción y explicaremos todas las herramientas y metodología empleada para realizar nuestro trabajo.

### 6.1. Introducción

#### 6.1.1. ¿Qué es DevOps?

*DevOps* es una metodología de trabajo focalizada en la comunicación, colaboración e integración entre desarrolladores de software y el resto de los profesionales TIC. El objetivo es ayudar a una organización a producir productos y servicios software más rápidamente, de mejor calidad y a un coste menor [48].

#### 6.1.2. ¿Por qué es tan importante?

Además de los esfuerzos por romper las barreras de comunicación y fomentar la colaboración entre los equipos de desarrollo y operaciones tecnológicas, uno de los principales valores de *DevOps* es lograr la satisfacción del cliente y prestar sus servicios en menos tiempo. *DevOps* también se ha creado para impulsar la innovación empresarial y ser el motor de continuas mejoras en los procesos [49].

*DevOps* permite ofrecer un mejor servicio cada vez, en menos tiempo, de mejor calidad y con mayor seguridad. Por ejemplo, puede reflejarse en la rapidez con la que llega al cliente

---

una nueva versión del producto o una nueva función manteniendo los mismos niveles de calidad y seguridad, o en el poco tiempo que se necesita para identificar un problema o un error y, a continuación, solucionarlo y volver a publicar una versión corregida [49].

Sin duda, todo este trabajo de *DevOps* se sustenta en una infraestructura subyacente con un rendimiento, una disponibilidad y una fiabilidad fluida y sin interrupciones del software, el cual se desarrolla y se prueba en primer lugar y, luego, se lanza a la fase de producción [49].

### 6.1.3. Ventajas

Aplicar la metodología *DevOps* supone muchos beneficios a la hora de trabajar y crear herramientas aplicando las metodologías *Agile*. Algunas de estas ventajas son [49]:

- Una mejor y más rápida entrega de productos
- Resolución de problemas en menos tiempo y con menor complejidad
- Mejor escalabilidad y disponibilidad
- Entornos de funcionamiento más estables
- Mejor utilización de los recursos
- Mayor automatización
- Mayor visibilidad de resultados del sistema
- Mayor innovación

### 6.1.4. Etapas del ciclo de DevOps

*DevOps* está compuesto las siguientes etapas: [49]:

- **Planificación:** Se definen los requisitos y valores empresariales.
- **Codificación:** Esta fase implica el diseño del software y la creación del código.
- **Compilación:** En esta fase se gestionan las versiones y las compilaciones del software, que utilizan herramientas automatizadas para ayudan a compilar y crear paquetes de código y publicarlos después para la producción. Se utilizan repositorios de código fuente o repositorios de paquetes que también empaquetan la infraestructura que se necesita para el lanzamiento del producto.
- **Prueba:** Esta fase incluye la realización de pruebas continuas (manuales o automatizadas) para garantizar la calidad de la programación.



- **Puesta en marcha.** En esta fase se emplean herramientas que ayudan a gestionar, coordinar, programar y automatizar las tareas de producción de las versiones de productos.
- **Funcionamiento.** En esta fase se gestiona el software durante su producción.
- **Supervisión.** En esta fase se identifica y recopila información sobre problemas que surgen en una versión de software específica que se encuentra en producción.

### 6.1.5. Prácticas de DevOps

Para aplicar de forma correcta la unión entre desarrollo y operaciones, deben llevarse a cabo unas prácticas específicas. Las siguientes son las más recomendadas para seguir de forma completa la metodología *Agile* [50]:

- **Integración continua:** Con esta práctica se busca combinar los cambios de código de manera habitual, para que automáticamente estos se actualicen y se ejecuten las pruebas pertinentes. De esta forma, se encuentran errores más rápidamente, se mejora la calidad del software y aumenta la velocidad de desarrollo a la hora de publicar nuevas versiones.
- **Entrega continua:** Se tratan otras prácticas en las cuales se compila, prueba y prepara de forma automática cualquier cambio en el código y es entregado a la fase de producción. Esto irá de la mano de la integración continua, de forma que podamos implementar nuevas versiones o cambios más rápidamente.
- **APIs o microservicios:** Las APIs nos sirven para crear pequeños servicios de software, de manera independiente, que se comunican con servicios o microservicios a través de una interfaz. Con todo esto logramos que sea más rápido desarrollar el conjunto global de nuestro software, puesto que las APIs son independientes y podemos trabajar sobre ellas sin que afecte al resto de microservicios.
- **Sistemas como código:** Se administra la infraestructura de sistemas aplicando técnicas de desarrollo de software con código, interactuando con ella mediante la programación. Al estar definidos por código, los servidores y demás sistemas se podrán implementar, nuevamente, de manera rápida aplicando patrones estandarizados.
- **Monitorización y registro:** Monitorizar el funcionamiento de una aplicación y realizar un registro de cómo el usuario se relaciona con dicha herramienta, será crucial para entender cómo funciona, dónde falla y qué habría que mejorar. Con esta práctica logramos que los ingenieros *DevOps* sean mucho más eficientes a la hora de realizar cambios en el software o el sistema.
- **Comunicación y colaboración:** Como ya hemos mencionado, la cultura *DevOps* tiene como base una buena comunicación y cooperación entre las partes de desarrollo y operaciones.

---

## 6.2. Azure DevOps

Para seguir la metodología definida por *DevOps*, hemos tomado la decisión de utilizar *Azure DevOps*, una plataforma de software como servicio o *SaaS* (*Software as a Service*) de Microsoft. Dicha plataforma nos proporciona una excelente opción para orquestar una cadena de herramientas *DevOps*, de un extremo a otro, para desarrollar e implementar software [51]. También se integra con la mayoría de las herramientas líderes en el mercado. Algunos de los servicios que ofrece son los siguientes [52]:

- **Azure Boards:** Herramientas *Agile* para que un equipo pueda planear su trabajo, debatir sobre él y hacer un seguimiento.
- **Azure Pipelines:** Servicio para compilar, probar e implementar código con integración y entrega continua que funciona con cualquier lenguaje, plataforma y nube.
- **Azure Repos:** Proporciona un número ilimitado de repositorios GIT hospedados en la nube, mediante los cuales podemos colaborar para compilar código de más calidad con solicitudes de incorporación de cambios y administración avanzada de archivos.
- **Azure Test Plans:** Servicio para probar y distribuir aplicaciones, de forma segura, utilizando herramientas de pruebas manuales y exploratorias.
- **Azure Artifacts:** Proporciona la creación y almacenamiento de paquetes, los cuales pueden ser compartidos con nuestro equipo e incorporados a canalizaciones (o *pipelines*) de CI/CD.

## 6.3. Integración continua

La integración continua o *CI* (*Continuous Integration*) es una práctica de desarrollo de software mediante la cual los desarrolladores combinan los cambios en el código en un repositorio central de forma periódica, tras lo cual se ejecutan versiones y pruebas automáticas. La integración continua se refiere en su mayoría a la fase de creación o integración del proceso de publicación de software y conlleva un componente de automatización (como por ejemplo, un servicio de versiones) y un componente cultural (como por ejemplo, aprender a integrar con frecuencia). Los objetivos clave de la integración continua consisten en encontrar y arreglar errores con mayor rapidez, mejorar la calidad del software y reducir el tiempo que se tarda en validar y publicar nuevas actualizaciones de software [53].

Antes de empezar, hemos creado un repositorio de código en *Azure Repos* que contendrá todo el código fuente de nuestro proyecto.

Una vez que tenemos el repositorio creado, vamos a crear un *pipeline* en *Azure Pipelines* con el proceso de integración continua. Dicho *pipeline* se ejecutará cada vez que se suba una nueva versión de nuestro código al repositorio y realizará las siguientes tareas:

1. Crear el plan de *Terraform* que se usará posteriormente para crear o actualizar la infraestructura en la nube. Esta tarea sólo se ejecutará cuando existan cambios en la infraestructura.
2. Compilar el proyecto de *Azure Functions* y generar un fichero comprimido, el cual se empleará para desplegar la aplicación de funciones en *Azure*.
3. Obtener del proyecto los artefactos necesarios para configurar el servicio *API Management* para nuestra *API REST*. Los artefactos que necesita son los siguientes:
  - (a) Fichero *JSON* con la configuración necesaria para implementar nuestra instancia de *API Management*.
  - (b) Fichero *JSON* con los valores de los parámetros empleados en el fichero *JSON* que contiene la configuración de *API Management*.
  - (c) Script de *PowerShell* que será ejecutado para crear o configurar el servicio de *API Management*.
4. Compilar la aplicación web desarrollada en *Blazor* para obtener los artefactos necesarios para desplegarla en el contenedor *blob* de nuestra cuenta de almacenamiento en *Microsoft Azure*.

## 6.4. Entrega continua

La entrega continua o *CD (Continuous Delivery)* es un enfoque de la ingeniería de software en el que los equipos producen software en ciclos cortos, lo que garantiza que el software se pueda lanzar de manera confiable en cualquier momento. Tiene como objetivo construir, probar y lanzar software con mayor velocidad y frecuencia. Este enfoque ayuda a reducir el costo, el tiempo y el riesgo de realizar cambios al permitir más actualizaciones incrementales para las aplicaciones en producción. Un proceso de implementación sencillo y repetible es importante para la entrega continua [54].

En nuestro proyecto de *Azure DevOps*, hemos creado un nuevo *pipeline* de tipo *release* que, ejecutándose tras el *pipeline* de integración continua, realizará las siguientes tareas:

1. Aplicar el plan de *Terraform*, en caso de existir, el cual actualizará la infraestructura en la nube.
2. Desplegar el proyecto de *Azure Functions* mediante el envío del fichero comprimido que fue creado durante la fase de integración continua.
3. Ejecutar el script que se encargará de crear o actualizar el proyecto de *API Gateway*.
4. Publicar los artefactos correspondientes a nuestra aplicación web en el contenedor *blob* de nuestra cuenta de almacenamiento en *Microsoft Azure*.

---

## 6.5. Infraestructura como servicio

La infraestructura como código o *IaC* (*Infrastructure as Code*) es la administración de la infraestructura (redes, máquinas virtuales, balanceadores de carga, topología de conexión, ...) en un modelo descriptivo, empleando el mismo control de versiones que utiliza el equipo de *DevOps* para el código fuente. Respetando el principio de que el mismo código fuente genera el mismo código binario, un modelo de *IaC* genera el mismo entorno cada vez que se aplica. *IaC* es una práctica clave de *DevOps* y se usa junto con la entrega continua [55].

Con todas las características que nos aporta la infraestructura como código, nos encontramos con los siguientes desafíos:

- Se necesita aprender a programar.
- Se desconoce el impacto del cambio.
- En ocasiones, es necesario revertir el cambio.
- No se pueden rastrear cambios.
- No se puede automatizar un recurso
- Múltiples entornos (diferentes proveedores) para la infraestructura.

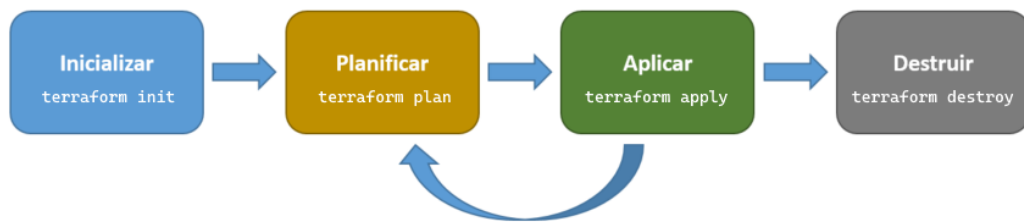
Para solucionar estos inconvenientes, vamos a emplear la herramienta *Terraform*.

*Terraform* es una herramienta para infraestructura como código desarrollada por *HashiCorp*. Se utiliza para definir y aprovisionar una infraestructura completa utilizando el lenguaje declarativo *HashiCorp Configuration Language* (*HCL*). Es muy similar a herramientas como *CloudFormation* de *Amazon Web Services* o *Azure Resource Manager* de *Microsoft* [56].

Las principales ventajas de *Terraform* son las siguientes [56]:

- Realiza orquestación, no sólo gestión de configuración.
- Admite múltiples proveedores como *AWS*, *Azure*, *GCP*, *DigitalOcean* y muchos más.
- Proporciona una infraestructura inmutable donde la configuración cambia sin problemas.
- Utiliza un lenguaje fácil de entender.
- Es fácil de portar a otro proveedor.

El ciclo de vida de *Terraform*, consiste en las fases: **init** (*inicializar*), **plan** (*planificar*), **apply** (*aplicar*) y **destroy** (*destruir*). En la Figura 6.1 podemos ver un esquema de dicho ciclo de vida. A continuación, vamos a detallar en qué consiste cada fase [56]:



**Figura 6.1:** Diagrama del ciclo de vida de Terraform

- *terraform init*: Inicializa el entorno *Terraform* de forma local. Por lo general, se ejecuta sólo una vez por sesión.
- *terraform plan*: Compara el estado de *Terraform* con el estado actual en la nube, creando y mostrando un plan de ejecución. Esta acción no cambia la implementación (solo lectura).
- *terraform apply*: Ejecuta el plan (actualiza la implementación en la nube).
- *terraform destroy*: Elimina todos los recursos que se rigen por el entorno de *Terraform*.



## 7. Experimentos y resultados

---

Una vez que tenemos el punto de conexión al modelo de predicción y el servicio de recomendación de playlists activos en nuestro proveedor de servicios en la nube, vamos a realizar una serie de experimentos para evaluar el desempeño del trabajo que hemos realizado. Para finalizar, evaluaremos nuestro sistema empleando una serie de métricas que utiliza *Spotify*.

### 7.1. Experimentos

Los experimentos que hemos seleccionado para evaluar el comportamiento de nuestro servicio, se basan en recomendaciones sobre playlists del conjunto de test, que cuentan con información incompleta (no se dispone del título, sólo conocemos las playlist que la conforman, ...). También vamos a evaluar el comportamiento de nuestro sistema en varias situaciones de *arranque el frío*. Por último, aunque no es una funcionalidad disponible actualmente en nuestro sistema, vamos a ver qué títulos, características de pistas, artistas o pistas son similares a los que le indiquemos.

#### 7.1.1. Recomendaciones sobre playlists del conjunto de test

Para este experimento, hemos elegido 3 playlists en las que hay parte de la información de la cual desconocemos:

1. Conocemos el título de la playlist pero no las pistas.
2. Conocemos el título y algunas de las pistas que conforman la playlist.
3. Desconocemos el título de la playlist pero disponemos de las pistas que la conforman.

---

Nombre	Álbum	Artista	Año
Bailando	SEX AND LOVE	Enrique Iglesias	2014
Vivir Mi Vida	3.0	Marc Anthony	2013
El Perdón	Fénix	Nicky Jam	2017
Despacito	Despacito (feat Daddy Yankee)	Luis Fonsi	2017
Danza Kuduro	Meet The Orphans	Don Omar	2010
DUELE EL CORAZON	DUELE EL CORAZON	Enrique Iglesias	2016
Darte un Beso	Soy el Mismo	Prince Royce	2013
Suavemente	Suavemente	Elvis Crespo	1998
Hasta el Amanecer	Fénix	Nicky Jam	2017
Corazon Sin Cara	Prince Royce	Prince Royce	2010

**Tabla 7.1:** Recomendación de la playlist *spanish playlist*

#### **Playlist 1000002: *spanish playlist* (sin pistas disponibles)**

En esta playlist desconocemos las pistas que la conforman, pero partiendo del título esperamos obtener canciones de artistas que cantan en español.

Si solicitamos a nuestro sistema que nos recomiende una playlist similar, obtenemos otra cuyas canciones principales se muestran en la Figura 7.1. Como podemos apreciar, las pistas se identifican perfectamente con el título, ya que todas ellas pertenecen a artistas que cantan en español.

#### **Playlist 1008130: *throwback songs* (10 pistas disponibles)**

Para esta playlist, conocemos 10 de las pistas que la conforman y su título, *throwback songs*. En la Tabla 7.2 podemos ver el detalle de las pistas de las cuales disponemos información. La temática de estas canciones es música pop internacional, de entre los años 2007 y 2015.

Como podemos observar en la Tabla 7.3, tanto los géneros, artistas y fechas de lanzamiento de las 10 primeras pistas de la playlist que se nos ha recomendado, se relacionan con el contenido de la playlist sobre la que hemos solicitado una recomendación.

#### **Playlist 1013678: Título desconocido y 10 pistas disponibles**

Aunque no disponemos del título de esta playlist, si observamos la Figura 7.4 y vemos las canciones por las que está compuesta, podemos concluir que es una playlist con música de tipo urbano (R&B, Hip-Hop, Rap, ...) y de artistas estadounidenses.



Nombre	Álbum	Artista	Año
Mirrors	The 20/20 Experience ...	Justin Timberlake	2013
Blame	Motion	Calvin Harris	2014
Baby	My Worlds	Justin Bieber	2010
Apologize	Dreaming Out Loud	OneRepublic	2007
Your Type	Emotion	Carly Rae Jepsen	2015
Explosions	Halcyon	Ellie Goulding	2012
DJ Got Us Fallin' In Love	Raymond v Raymond ...	Usher	2010
Somebody To Love	My Worlds	Justin Bieber	2010
Don't Dream It's Over	Glee: The Music ...	Glee Cast	2014
You'll Always Find ...	Hannah Montana The ...	Hannah Montana	2009

**Tabla 7.2:** Contenido de la playlist *throwback songs*

Nombre	Álbum	Artista	Año
He Could Be the One	Hannah Montana 3	Hannah Montana	2009
Teenage Dream	Teenage Dream	Katy Perry	2010
Beggin' On Your Knees	Victorious: Music From ...	Victorious Cast	2010
Ordinary Girl	Hannah Montana Forever	Hannah Montana	2010
Best Friend's Brother	Victorious: Music From ...	Victorious Cast	2010
I Wanna Know You	Hannah Montana 3	Hannah Montana	2009
Telephone	The Fame Monster	Lady Gaga	2009
Bad Romance	The Fame Monster	Lady Gaga	2009
Freak The Freak Out	Victorious: Music From ...	Victorious Cast	2010
You'll Always Find ...	Hannah Montana The ...	Hannah Montana	2009

**Tabla 7.3:** Recomendación sobre la playlist *throwback songs*

Nombre	Álbum	Artista	Año
Hard In Da Paint	Flockaveli	Waka Flocka Flame	2010
No Hands ...	Flockaveli	Waka Flocka Flame	2010
Scholarship	Stay Trippy	Juicy J	2013
Wild for the Night	LONG.LIVE.A\$AP ...	A\$AP Rocky	2013
Work REMIX	Trap Lord	A\$AP Ferg	2013
All Gold Everything ...	Don't Be S.A.F.E.	Trinidad James	2013
Believe It ...	Dreams and Nightmares	Meek Mill	2012
No Lie	Based On A T.R.U. Story	2 Chainz	2012
The Next Episode	2001	Dr. Dre	1999
Smokin & Drinkin	Old	Danny Brown	2013

**Tabla 7.4:** Contenido de la playlist #1013678

---

Nombre	Álbum	Artista	Año
Turn Down for What	Turn Down for What	DJ Snake	2013
All Gold Everything - Remix	Don't Be S.A.F.E.	Trinidad James	2013
Pop That	Excuse My French	French Montana	2013
Watch Out	Watch Out	2 Chainz	2015
We Dem Boyz	Blacc Hollywood	Wiz Khalifa	2014
Who Do You Love?	My Krazy Life	YG	2014
Where Ya At	DS2	Future	2015
0 To 100 / The Catch Up	0 To 100 / The Catch Up	Drake	2013
Jumpman	What A Time To Be Alive	Drake	2015
Birthday Song	Based On A T.R.U. Story	2 Chainz	2012

**Tabla 7.5:** Recomendación sobre la playlist #1013678

Una vez realizada la recomendación para una nueva playlist, cuyas 10 primeras canciones podemos ver en la Figura 7.5, vemos que las pistas que la conforman pertenecen a los mismos géneros y también a artistas estadounidenses.

### 7.1.2. Arranque en frío

El siguiente experimento está basado en el problema del arranque en frío, trataremos de recomendar una playlist sobre otra que no se encuentra en el sistema. Este caso es diferente al del primer experimento, ya que en dicho caso la playlist se encontraba almacenada en el sistema pero no tenía pistas asignadas.

La primera recomendación que vamos a solicitar es sobre una playlist ficticia llamada *The Disney Playlist*, de la cual esperamos obtener canciones que pertenezcan a dicha temática.

Si observamos la Tabla 7.6, que contiene las 10 primeras canciones de la playlist que nos ha sugerido el sistema, podemos apreciar que todas las canciones se corresponden con películas o series de la factoría *Disney*.

Por último, vamos a realizar una última recomendación sobre otra playlist ficticia titulada *Christmas 2021*, de la cual esperamos obtener otra playlist que contenga canciones navideñas.

Si nos fijamos en la Tabla 7.7, que contiene las 10 primeras pistas de la playlist generada, podemos comprobar que todas las canciones son de carácter navideño.

Nombre	Álbum	Artista	Año
A Whole New World	Aladdin	Lea Salonga	1992
Hakuna Matata	The Lion King	Nathan Lane	1994
I'll Make a Man Out of You	Mulan	Donny Osmond	1998
Under the Sea	Little Mermaid	Samuel E. Wright	2006
Colors Of The Wind	Pocahontas	Judy Kuhn	1995
I Won't Say (I'm in Love)	Hercules Original Soundtrack	Lillias White	1997
Kiss the Girl	Little Mermaid	Samuel E. Wright	2006
Reflection	Mulan	Lea Salonga	1998
Circle Of Life	The Lion King	Carmen Twillie	1994
Be Our Guest	Beauty and the Beast	Angela Lansbury	1991

**Tabla 7.6:** Recomendación de playlist para el título "The Disney Playlist"

Nombre	Álbum	Artista	Año
All I Want for Christmas Is You	Merry Christmas	Mariah Carey	1994
Rockin' Around The Christmas Tree	Anthology 1956-1980	Brenda Lee	1991
It's Beginning To Look A Lot Like ...	Christmas	Michael Bublé	2011
It's the Most Wonderful Time of ...	The Andy Williams ...	Andy Williams	1963
Holly Jolly Christmas	Christmas	Michael Bublé	2011
Jingle Bell Rock	Mercy Me ...	Anita Kerr Singers	2017
A Holly Jolly Christmas	Have A Holly ...	Burl Ives	1965
White Christmas	Holiday Inn	Bing Crosby	1942
Blue Christmas	Elvis' Christmas Album	Elvis Presley	1957
White Christmas (feat. Shania Twain)	Christmas	Michael Bublé	2011

**Tabla 7.7:** Recomendación de playlist para el título "Christmas 2021"

---

### 7.1.3. Obtención de etiquetas similares

Una posibilidad que ofrece nuestro modelo de recomendación es la de buscar características de playlists que sean similares a una dada. En este caso, hemos realizado una prueba con términos similares *latino* y *running*. Los resultados que hemos obtenido son los siguientes:

- **chill:** chillin, vibes, lowkey, relax, chill, calm, chillax, mellow, mood, relaxed, breathe, snooze.
- **running:** run, marathon, workout, cardio, gym, runnin, sweat, exercise, race, workouts, crossfit, spin.

Como podemos apreciar, los resultados obtenidos para cada término guardan relación entre ellos.

## 7.2. Resultados

Aunque la librería *LightFM* proporciona algunas métricas con las que evaluar el resultado de nuestro modelo, no ha sido posible obtenerlos debido al gran volumen de datos con el que hemos trabajado y al saldo limitado que tenemos en nuestra cuenta de *Microsoft Azure*.

Tras tener la instancia de cómputo durante más de 6 horas en ejecución, la librería *LightFM* no ha proporcionado ningún resultado, por lo que hemos tomado la decisión de detener la instancia y buscar otra forma con la que podamos evaluar nuestro modelo.

Como comentamos en el Capítulo 4, recientemente se ha liberado el conjunto *Million Playlist Dataset* [34], construido por *Spotify* para la competición *RecSys Challenge* (edición del 2018) [33]. Este conjunto puede ser obtenido en el portal *Alcroud* [57], pudiendo participar en el reto abierto *Spotify Million Playlist Dataset Challenge*. Como en el reto que está disponible, tras enviar los resultados de las predicciones sobre el conjunto de test, nos ofrece una serie de métricas y podemos comparar los resultados con los de otros participantes, hemos decidido re-entrenar el modelo empleando dicho conjunto (ya que el conjunto que creamos en su momento está basado en el *Million Playlist Dataset*).

### 7.2.1. Definición de métricas

Los envíos realizados al reto se evalúan utilizando las siguientes métricas que detallamos en este apartado. Todas las métricas son evaluadas tanto a nivel de pista (coincidencia exacta de pista) como a nivel de artista (cualquier pista del mismo artista es una coincidencia).

A continuación, denotamos el conjunto de pistas reales (las que debemos recomendar) por **G** y la lista ordenada de pistas recomendadas por **R**. El tamaño de un conjunto o lista se denota por  $|\cdot|$ , y usamos *from:to-subscripts* para indexar una lista.

### R-Precision

*R-precision* es el número de pistas relevantes recuperadas dividido por el número de pistas relevantes conocidas (es decir, la cantidad de pistas retenidas):

$$R - precision = \frac{|G \cap R_{1:|G|}|}{|G|}$$

La métrica se promedia en todas las listas de reproducción del conjunto de test (o desafío). Esta métrica recompensa el número total de pistas relevantes recuperadas (independientemente del orden).

### Normalized Discounted Cumulative Gain (NDCG)

*Discounted Cumulative Gain*, o *DCG*, mide la calidad de clasificación de las pistas recomendadas y aumenta cuando las pistas relevantes se colocan más arriba en la lista. La *DCG* normalizada (*NDCG*) se determina calculando la *DCG* y dividiéndola por la *DCG* ideal en la que las pistas recomendadas están perfectamente clasificadas:

$$DCG = rel_1 + \sum_{i=2}^{|R|} \frac{rel_i}{\log_2 i}$$

donde  $rel_i$  es la relevancia obtenida de la pista que se encuentra en la posición  $i$ . El *DCG* ideal o *IDCG* es, en nuestro caso, igual a:

$$IDCG = 1 + \sum_{i=2}^{|G \cap R|} \frac{1}{\log_2 i}$$

Si el tamaño de la intersección establecida de **G** y **R** es vacío, entonces el *IDCG* es igual a 0. La métrica *NDCG* ahora se calcula como:

$$NDCG = \frac{DCG}{IDCG}$$

---

## Recommended song clicks

*Recommended song clicks* es una función de *Spotify* que, dado un conjunto de pistas en una lista de reproducción, recomienda 10 pistas para agregar a la lista de reproducción. La lista se puede actualizar para producir 10 resultados más. Los clics de canciones recomendadas son la cantidad de actualizaciones necesarias antes de que se encuentre una pista relevante. El número máximo de clics que se ha establecido es de 50. En los casos donde el valor sea superior, se considera que la métrica no está disponible, es decir, no hay pistas relevantes en **R**. La fórmula para calcular este valor es:

$$clicks = \left\lceil \frac{\arg \min_i \{R_i : R_i \in G\} - 1}{10} \right\rceil$$

### 7.2.2. Métricas obtenidas

Una vez que hemos realizado las predicciones para las playlists del reto y las hemos enviado al sistema de evaluación, hemos obtenido los siguientes resultados:

- **R-prec:** 0,220
- **NDCG:** 0,341
- **Recommended Song Clicks:** 2,212

También hemos enviado los resultados obtenidos tras emplear el conjunto de datos sobre el proyecto *Generación automática de playlist de canciones mediante técnicas de minería de datos* [11] al reto de *Spotify* en *Alcroud* [57], de tal forma que podamos comprobar cómo ha progresado nuestro trabajo <sup>1</sup>. Si comparamos los resultados con los del primer clasificado (a fecha 21/08/2021), y que mostramos en la Tabla 7.8, podemos ver que los resultados obtenidos con el modelo entrenado en éste trabajo son bastante prometedores.

Posición	R-prec	NDCG	Recommended Song Clicks	Comentario
#1	0,220	0,386	1,932	Primer clasificado
...	...	...	...	
#5	0,187	0,341	2,212	Modelo Serendipity (TFM)
...	...	...	...	
#27	0.096	0.189	8.289	Modelo obtenido con TFG

**Tabla 7.8:** Resultados obtenidos en la competición de AICrowd

---

<sup>1</sup>Debido a un error con la cuenta empleada, el nombre de usuario aparece como *Unknown User*

## 8. Conclusiones y propuestas

---

En este capítulo vamos a hablar sobre las conclusiones que hemos obtenido tras haber entrenado el modelo de recomendación y desplegado un sistema de recomendación de playlists. Para finalizar, mencionaremos algunas mejoras que hemos planteado para realizar en el futuro, junto a otras líneas de investigación en el ámbito de los sistemas de recomendación musical en las cuales nos gustaría profundizar para incorporarlas también a nuestro trabajo.

### 8.1. Conclusiones

Durante la realización de este Trabajo de Fin de Máster, hemos podido comprobar el gran esfuerzo que requiere el desarrollo de un modelo de recomendación. A la hora de elegir qué características han de usarse para identificar las pistas y las playlists, hemos tenido que realizar numerosos entrenamientos para el modelo y ejecutar un gran número de pruebas con las que determinar qué configuración era la mas adecuada para este problema.

Una vez obtenido el modelo de recomendación, uno de los principales retos que nos hemos encontrado ha sido el de su puesta en funcionamiento (fase de producción). Al haber obtenido un modelo de un tamaño considerable, hemos tenido que probar numerosas alternativas para ver cuál era la más apropiada teniendo en cuenta el tiempo de respuesta para obtener los resultados y su coste económico.

También cabe resaltar el tiempo empleado en montar toda la infraestructura necesaria para el funcionamiento del sistema de recomendación, en la que hemos tenido que utilizar numerosos servicios y buscar un diseño en el que obtuviéramos unos tiempos de ejecución razonables sin que repercutieran de forma negativa en la factura mensual del proveedor de servicios en la nube.

Con el sistema de recomendación en funcionamiento, tras realizar numerosas pruebas y

---

solicitar la participación de otras personas para que evaluaran los resultados que se ofrecían, podemos decir que estamos muy satisfechos con los resultados obtenidos. Todas las playlists que han sido recomendadas, contenían un número elevado de canciones que estaban relacionadas con lo que el usuario esperaba.

## 8.2. Propuestas

Para finalizar este capítulo, vamos a exponer una serie de propuestas que queremos realizar en un futuro para mejorar nuestro sistema de recomendación de playlists, así como las técnicas que podríamos usar para implementar dichas propuestas.

### 8.2.1. Funcionalidad del sistema

#### Evaluación de las recomendaciones ofrecidas

Una de las principales funcionalidades que queremos ofrecer en el futuro, es la de que un usuario pueda ofrecer *feedback* sobre los resultados propuestos. Algunas formas de desarrollar esta funcionalidad pueden ser la del empleo de *me gusta/no me gusta*, o dar la opción de eliminar aquellas pistas de las playlists que no resulten de su agrado, almacenando toda esta información para futuros entrenamientos del modelo.

#### Recomendaciones de playlists mediante el uso de asistentes

Con esta funcionalidad queremos ofrecer al usuario recomendaciones a través de asistentes, como *Google Assistant* o *Alexa*. El usuario debe de invocar al asistente, decirle el título de la playlist y una descripción del contenido que espera que contenga, interpretar esa petición, con el resultado añadirla a su cuenta de *Spotify* y reproducirla en el dispositivo empleado. Un ejemplo de invocación sería el siguiente:

*Ok Google, créame una playlist titulada Throwbacks que contenga canciones de Britney Spears, Jennifer López y Beyoncé que pertenezcan a la década de los 2000.*

#### Re-entrenamiento con nueva información

Como nuestro sistema va a recibir playlists que contienen canciones desconocidas o novedades, y también se enfrentará a títulos con los que nunca ha trabajado, queremos ser capaces de recopilar estas nuevas playlist (con permiso del usuario) y emplearlas en futuros entrenamientos.



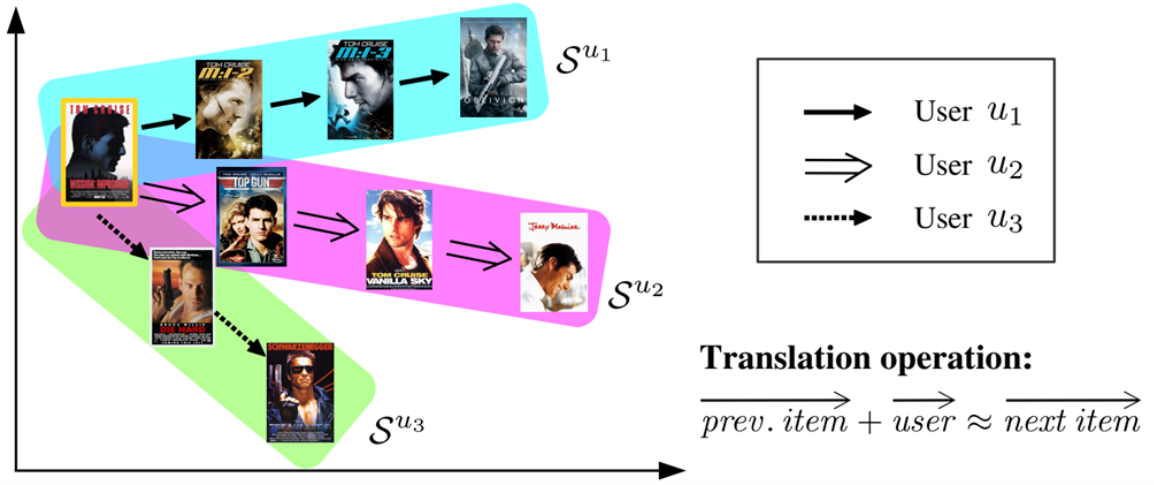


Figura 8.1: Ejemplo de modelo secuencial

También queremos que nuestro sistema sea capaz de determinar qué playlists de las que se encuentran almacenadas no se están usando para producir recomendaciones y eliminarlas, con lo que reducir el tamaño del modelo y el tiempo empleado en entrenarlo.

### 8.2.2. Líneas de investigación

#### Características de audio

Aunque en un principio intentamos emplear las características de audio que ofrece *Spotify* para las canciones de su catálogo, no conseguimos entrenar un modelo de recomendación que hiciera uso de dichas características. En el futuro, nos gustaría investigar el motivo por el cual nuestro modelo no fue capaz de usar las características de audio para realizar el entrenamiento, ya que consideramos que esta información es de gran valor y aporta información con la que podemos mejorar las recomendaciones de playlists.

#### Modelos secuenciales

En nuestro trabajo, no hemos tenido en cuenta la posición en las que se encuentran las pistas de las playlists de nuestro conjunto, al igual que tampoco hemos establecido un criterio a la hora de ofrecer las recomendaciones a las playlists empleando la posición.

Recientemente se está experimentando con técnicas que permiten realizar recomendaciones siguiendo la secuencia de interacción del usuario, como por ejemplo *Translation-Based Recommendation* [58]. Si tomamos como ejemplo la secuencia de visionado en un

---

servicio de streaming de vídeo como *Netflix*, podemos recomendar a los usuarios qué película pueden visionar a continuación según las secuencias de interacción que hemos obtenido de otros usuarios.

En la Figura 8.1 podemos ver un ejemplo de recomendación mediante esta técnica para la película *Misión Imposible*.

## A. Contenido del CD

---

En el CD que acompaña a esta memoria, podemos encontrar los siguientes contenidos:

1. Memoria en formato *PDF*
2. Código empleado para la realización de este trabajo: libretas de *Jupyter*, proyecto de funciones, scripts, aplicación web, ficheros para *Terraform* (infraestructura) y el fichero de requisitos con las bibliotecas externas de las que hacemos uso.
3. Libretas *Jupyter* en formato *PDF*.
4. Vídeo con una demostración del uso de la aplicación web.



## Referencias bibliográficas

---

- [1] M. Schedl, H. Zamani, C.-W. Chen, Y. Deldjoo, and M. Elahi, "Current challenges and visions in music recommender systems research," Abril 2018. [En línea]. Disponible en: <https://link.springer.com/article/10.1007/s13735-018-0154-2> [con acceso el 02-09-2021].
- [2] P. Adamopoulos and A. Tuzhilin, "On Unexpectedness in Recommender Systems: Or How to Better Expect the Unexpected," Diciembre 2014. [En línea]. Disponible en: <https://dl.acm.org/doi/abs/10.1145/2559952> [con acceso el 03-09-2021].
- [3] G. Adomavicius, B. Mobasher, F. Ricci, and A. Tuzhilin, "Context-aware recommender systems," 2011. [En línea]. Disponible en: <https://ojs.aaai.org//index.php/aimagazine/article/view/2364> [con acceso el 03-09-2021].
- [4] F. B, S. M, and T. M, "Personality and emotional states: understanding users music listening needs," 2015. [En línea]. Disponible en: [http://phenicx.upf.edu/system/files/publications/umap2015\\_submission\\_3.pdf](http://phenicx.upf.edu/system/files/publications/umap2015_submission_3.pdf) [con acceso el 03-09-2021].
- [5] P. J. Rentfrow and S. D. Gosling, "The do re mi's of everyday life: The structure and personality correlates of music preferences," 2013. [En línea]. Disponible en: <https://doi.org/10.1037/0022-3514.84.6.1236> [con acceso el 03-09-2021].
- [6] M. Gillhofer and M. Schedl, "Iron maiden while jogging, debussy for dinner? an analysis of music listening behavior in context," 2015. [En línea]. Disponible en: <http://phenicx.upf.edu/node/182.html> [con acceso el 03-09-2021].
- [7] Y. W. Xinxu Wang, David Rosenblum, "Context-aware mobile music recommendation for daily activities," 2015. [En línea]. Disponible en: <https://projet.liris.cnrs.fr/imagine/pub/proceedings/ACM-MULTIMEDIA-2012/mm/p99.pdf> [con acceso el 03-09-2021].

- 
- [8] M. Kaminskas, F. Ricci, and M. Schedl, "Hypergraph models of playlist dialects," Octubre 2013. [En línea]. Disponible en: <https://dl.acm.org/doi/10.1145/2507157.2507180> [con acceso el 03-09-2021].
- [9] B. McFee and G. Lanckriet, "Location-aware music recommendation using auto-tagging and hybrid matching," 2012. [En línea]. Disponible en: [https://ismir2012.ismir.net/event/papers/343\\_ISMIR\\_2012.pdf](https://ismir2012.ismir.net/event/papers/343_ISMIR_2012.pdf) [con acceso el 03-09-2021].
- [10] E. Zheleva, J. Guiver, E. Mendes Rodrigues, and N. Milić-Frayling, "Statistical Models of Music-listening Sessions in Social Media," 2010. [En línea]. Disponible en: <https://paginas.fe.up.pt/~eduarda/papers/wfp0858-zheleva.pdf> [con acceso el 03-09-2021].
- [11] M. Ángel Cantero Villora, "Generación automática de playlist de canciones mediante técnicas de minería de datos," Febrero 2020. [En línea]. Disponible en: <https://github.com/miguelangelcv/tfg-generacionplaylists> [con acceso el 12-08-2021].
- [12] U. de Castilla La Mancha, "Máster Universitario en Ingeniería Informática," 2020. [En línea]. Disponible en: <https://www.uclm.es/estudios/masteres/master-ingenieria-informatica-albacete> [con acceso el 08-09-2021].
- [13] S. Huang, "Introduction to Recommender System. Part 1.," Enero 2019. [En línea]. Disponible en: <https://hackernoon.com/introduction-to-recommender-system-part-1-collaborative-filtering-singular-value> [con acceso el 03-09-2021].
- [14] Tryolabs, "Introduction to Recommender Systems in 2019," Mayo 2018. [En línea]. Disponible en: <https://tryolabs.com/blog/introduction-to-recommender-systems/> [con acceso el 03-09-2021].
- [15] E. Grimaldi, "How to build a content-based movie recommender system with Natural Language Processing," Octubre 2018. [En línea]. Disponible en: <https://towardsdatascience.com/how-to-build-from-scratch-a-content-based-movie-recommender-with-natural-language-processing/> [con acceso el 03-09-2021].
- [16] R. Channel, "Hybrid recommendation approaches," Agosto 2017. [En línea]. Disponible en: <https://slideplayer.com/slide/4169010/> [con acceso el 03-09-2021].
- [17] "MagicPlaylist." [En línea]. Disponible en: <https://magicplaylist.co> [con acceso el 03-09-2021].
- [18] M. Almeida, "Playlist Generator." [En línea]. Disponible en: <https://homepages.dcc.ufmg.br/~marcos.almeida/playlistgenerator/index.html> [con acceso el 03-09-2021].
-

- [19] "Spotallike." [En línea]. Disponible en: <https://spotallike.com/> [con acceso el 03-09-2021].
- [20] J. V. Román, "CRISP-DM: La metodología para poner orden en los proyectos," Agosto 2016. [En línea]. Disponible en: <https://www.sngular.com/es/data-science-crisp-dm-metodologia> [con acceso el 03-08-2021].
- [21] A. Roy, "Introduction to CRISP-DM Framework for Data Science and Machine Learning," Julio 2018. [En línea]. Disponible en: <https://www.linkedin.com/pulse/chapter-1-introduction-crisp-dm-framework-data-science-anshul-roy> [con acceso el 03-08-2021].
- [22] A. Goicochea, "CRISP-DM, Una metodología para proyectos de Minería de Datos," Agosto 2009. [En línea]. Disponible en: <https://anibalgoicochea.com/2009/08/11/crisp-dm-una-metodologia-para-proyectos-de-mineria-de-datos> [con acceso el 03-08-2021].
- [23] W. Vorhies, "CRISP-DM, a standard methodology to ensure a good outcome," Julio 2009. [En línea]. Disponible en: <https://www.datasciencecentral.com/profiles/blogs/crisp-dm-a-standard-methodology-to-ensure-a-good-outcome> [con acceso el 03-08-2021].
- [24] S. A. Yazzi, "Una experiencia práctica de Scrum a través del aprendizaje basado en proyectos mediado por TIC en un equipo distribuido," Junio 2011. [En línea]. Disponible en: <https://core.ac.uk/download/pdf/9523302.pdf> [con acceso el 04-08-2021].
- [25] ProyectosAgiles.org, "Qué es scrum." [En línea]. Disponible en: <https://proyectosagiles.org/que-es-scrum/> [con acceso el 04-08-2021].
- [26] Conectart, "La metodología SCRUM. qué es y cómo funciona." [En línea]. Disponible en: <https://blog.conectart.com/la-metodologia-scrum-scrum-methodology/> [con acceso el 04-08-2021].
- [27] Z. Team, "Scrum 101: An Introduction To Scrum Project Management," Agosto 2018. [En línea]. Disponible en: <https://zenkit.com/en/blog/scrum-101-an-introduction-to-scrum-project-management/> [con acceso el 04-08-2021].
- [28] A. Scrum, "¿Qué es Scrum?." [En línea]. Disponible en: <https://www.agilescrum.cl/post/que-es-scrum> [con acceso el 04-08-2021].
- [29] J. A. F. Guerrero, "Definición y propuesta de un modelo para facilitar la transformación digital en las organizaciones," Julio 2020. [En línea]. Disponible en: <https://ruidera.uclm.es/xmlui/handle/10578/25558> [con acceso el 04-08-2021].

- 
- [30] J. A. Ochoa, "Pilares en la guía Scrum," Julio 2020. [En línea]. Disponible en: <https://castor.com.co/guia-scrum-2017-algunas-reflexiones-parte-ii/> [con acceso el 04-08-2021].
- [31] A. Abdollahzadeh, "CRISP-DM and Agile-Scrum methodology for Data Science Project Delivery," Mayo 2021. [En línea]. Disponible en: <https://www.linkedin.com/pulse/crisp-dm-agile-scrum-methodology-data-science-project-abdollahzadeh> [con acceso el 04-08-2021].
- [32] K. Johnson, "Github: Python and TypeScript gain popularity among programming languages," Diciembre 2020. [En línea]. Disponible en: <https://venturebeat.com/2020/12/02/github-python-and-typescript-gain-popularity-among-programming-languages> [con acceso el 04-08-2021].
- [33] X. Amatriain, H. Corona, M. Ekstrand, J. O'Donovan, and S. Pera, "12th ACM Conference on Recommender System," 2018. [En línea]. Disponible en: <https://recsys.acm.org/recsys18/> [con acceso el 12-08-2021].
- [34] C.-W. Chen, "The Million Playlist Dataset... Remastered," Diciembre 2020. [En línea]. Disponible en: <https://research.atspotify.com/the-million-playlist-dataset-remastered> [con acceso el 12-08-2021].
- [35] L. de la Ossa, "Análisis de opinión (sentiment analysis) en textos," Junio 2021. [En línea]. Disponible en: <https://github.com/cidaen/mc1-m8-textmining> [con acceso el 04-08-2021].
- [36] M. Kula, "LightFM Documentation," 2020. [En línea]. Disponible en: <https://lyst.github.io/lightfm/docs/index.html> [con acceso el 13-08-2021].
- [37] M. Kula, "Metadata Embeddings for User and Item Cold-start Recommendations," Julio 2015. [En línea]. Disponible en: <https://arxiv.org/pdf/1507.08439.pdf> [con acceso el 13-08-2021].
- [38] E. Rosenthal, "Learning to Rank Sketchfab Models with LightFM," Noviembre 2016. [En línea]. Disponible en: <https://www.ethanrosenthal.com/2016/11/07/implicit-mf-part-2> [con acceso el 13-08-2021].
- [39] Microsoft, "Informática sin servidor," 2021. [En línea]. Disponible en: <https://azure.microsoft.com/es-es/overview/serverless-computing/> [con acceso el 03-09-2021].
- [40] L. García, "¿Qué es Azure Cosmos DB?," Febrero 2018. [En línea]. Disponible en: <https://unpocodejava.com/2018/02/22/que-es-azure-cosmos-db/> [con acceso el 03-09-2021].
-



- [41] Microsoft, "Introducción a Azure Cosmos DB," Junio 2021. [En línea]. Disponible en: <https://docs.microsoft.com/es-es/azure/cosmos-db/introduction> [con acceso el 03-09-2021].
- [42] D. Singu, "Introduction to Cosmos DB," Mayo 2021. [En línea]. Disponible en: <https://medium.com/globant/introduction-to-cosmos-db-8d106bb7207> [con acceso el 03-09-2021].
- [43] Microsoft, "Introducción a Azure Functions," Noviembre 2020. [En línea]. Disponible en: <https://docs.microsoft.com/es-es/azure/azure-functions/functions-overview> [con acceso el 03-09-2021].
- [44] C. D. Alcolea, "Introducción a Azure Functions," Septiembre 2020. [En línea]. Disponible en: <https://openwebinars.net/blog/introduccion-azure-functions/> [con acceso el 03-09-2021].
- [45] L. López, "Introducción a Azure Functions," Mayo 2018. [En línea]. Disponible en: <https://medium.com/@lopezlucas/introduccion-a-azure-functions-f77a6b017847> [con acceso el 03-09-2021].
- [46] educative, "What is a REST API?," 2021. [En línea]. Disponible en: <https://www.educative.io/courses/web-application-software-architecture-101/qADAzX6yorR/> [con acceso el 04-09-2021].
- [47] J. L. Gonzáles, "Introducción a Azure API Management," Junio 2016. [En línea]. Disponible en: <https://www.compartimoss.com/revistas/numero-28/introduccion-a-azure-api-management/> [con acceso el 04-09-2021].
- [48] R. Digital, "¿qué es DevOps y que debes saber para convertirte en ello?," Mayo 2019. [En línea]. Disponible en: [http://www.rrhhdigital.com/secciones/tecnologia-e-innovacion/136859/Que-es-DevOps-y-que-debes-saber-para-convertirte-en-ello?target=\\_self](http://www.rrhhdigital.com/secciones/tecnologia-e-innovacion/136859/Que-es-DevOps-y-que-debes-saber-para-convertirte-en-ello?target=_self) [con acceso el 05-09-2021].
- [49] N. Web, "¿qué es DevOps?," 2021. [En línea]. Disponible en: <https://www.netapp.com/es/devops-solutions/what-is-devops/> [con acceso el 05-09-2021].
- [50] Ángel Mesones, "¿qué es la integración continua?," Septiembre 2020. [En línea]. Disponible en: <https://www.incentro.com/es-es/blog/stories/que-es-devops/> [con acceso el 05-09-2021].
- [51] Microsoft, "Azure DevOps," 2021. [En línea]. Disponible en: <https://azure.microsoft.com/es-es/services/devops/> [con acceso el 05-09-2021].
- [52] DevOpsGroup, "What is azure devops?," 2021. [En línea]. Disponible en: <https://www.devopsgroup.com/insights/resources/tutorials/all/what-is-azure-devops/> [con acceso el 05-09-2021].

- 
- [53] A. W. Services, “¿qué es la integración continua?,” 2021. [En línea]. Disponible en: <https://aws.amazon.com/es/devops/continuous-integration> [con acceso el 05-09-2021].
- [54] Ciberninjas, “¿qué es la integración continua?,” Septiembre 2021. [En línea]. Disponible en: <https://ciberninjas.com/implementacion-continua> [con acceso el 05-09-2021].
- [55] Microsoft, “¿qué es la infraestructura como código?,” Mayo 2021. [En línea]. Disponible en: <https://docs.microsoft.com/es-es/devops/deliver/what-is-infrastructure-as-code> [con acceso el 05-09-2021].
- [56] S. Singh, “Terraform Beginner’s Guide: Everything You Should Know,” Noviembre 2020. [En línea]. Disponible en: <https://k21academy.com/terraform-iac/terraform-beginners-guide/> [con acceso el 05-09-2021].
- [57] Spotify, “Spotify Million Playlist Dataset Challenge,” 2020. [En línea]. Disponible en: <https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge> [con acceso el 03-09-2021].
- [58] R. He, W.-C. Kang, and J. McAuley, “Translation-based recommendation,” Agosto 2017. [En línea]. Disponible en: <https://arxiv.org/pdf/1707.02410.pdf> [con acceso el 04-09-2021].