Universidad Autónoma de Madrid

Escuela Politécnica Superior

25|26

Trabajo fin de grado

Book-to-Soundtrack: Implementación de un Sistema de Recomendación de Música para Lectores

Marina López-Serrano Tapia

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\ Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC+

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática y Matemáticas

TRABAJO FIN DE GRADO

Book-to-Soundtrack: Implementación de un Sistema de Recomendación de Música para Lectores

Autor: Marina López-Serrano Tapia
Tutor: Alejandro Bellogín Kouki

mayo 2026

Todos los derechos reservados.
Queda prohibida, salvo excepción prevista en la Ley, cualquier
forma de reproducción, distribución comunicación pública
y transformación de esta obra sin contar con la autorización
de los titulares de la propiedad intelectual.
La infracción de los derechos mencionados puede ser constitutiva de
delito contra la propiedad intelectual *(arts. 270 y sgts. del Código Penal)*.

DERECHOS RESERVADOS
© 20 de Mayo de 2026 por UNIVERSIDAD AUTÓNOMA DE MADRID
Francisco Tomás y Valiente, n^o 1
Madrid, 28049
Spain

Marina López-Serrano Tapia
Book-to-Soundtrack: Implementación de un Sistema de Recomendación de Música para Lectores

Marina López-Serrano Tapia
C\ Marcelo Usera, 168

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A las personas con muchas ideas
y pocas ejecuciones

Toda mi alma se espació en tu alma,
y en ella viose como en claro espejo.

Pedro Antonio de Alarcón

Agradecimientos

Lo primero de todo, gracias a mi increíble tutor, Alejandro Bellogín Kouki, quien ha tenido un seguimiento insuperable conmigo y con el proyecto, y siempre ha hecho sugerencias muy útiles. Gracias por llevar este proyecto a buen puerto.

Gracias a los sonidos que, de forma omnipresente, suenan en mis cascos mientras trabajo. Sois mi protección y, también, la mejor ayuda para la concentración que tengo. No podría entrar en trance sin vosotros. Destaco, en el apartado de lo-fi: One Wayne G de Mac DeMarco y el canal de Youtube Yellow Cherry Jam. En el apartado de los audios ambientales que duran 10 horas: sonidos de lluvia, truenos y sonidos del mar.

Gracias a todes mis amigues por vuestra presencia en mi vida. Con eso es suficiente e importa más de lo que puedo expresar. Quería mencionar, personalmente, a mis amigues Katia, Bea y David por su entusiasmo por esta boba herramienta que he desarrollado; me recordáis por qué me decidí por este tema y no por otro más genérico. Gracias por recordarme lo que tengo dentro.

Gracias a la literatura. Concretamente, a la narrativa de ficción. Te he redescubierto en estos últimos meses y me he enamorado de nuevo. Me ayudas a conectar con partes de mí que creía enterradas, y me sorprendes una y otra vez con tu insistencia en quedarte en mi vida. Intentaré no dejarte escapar de nuevo.

Por último, a Kradno. Gracias por enseñarme lo que significa descansar. Lo que significa estar, sin ir a ningún sitio o pensar en hacerlo. Gracias por alcanzarme en un sitio que desde siempre he sentido que nadie estaría interesade en alcanzar. En palabras de Taylor Swift, 'You were bigger than the whole sky'.

Gracias.

Resumen

Hay una relación entre la literatura y la música que, aunque sea difícil de explicar, existe de forma implícita. El objetivo de este trabajo es hacer lo implícito explícito, mediante la creación de un algoritmo (llamado 'Book-to-Soundtrack') que reciba como entrada un libro y devuelva una recomendación de música. Es un trabajo adyacente a los sistemas de recomendación y toma conocimiento de la literatura existente, aunque no es el mismo concepto.

Para la consecución adecuada del algoritmo será necesaria la creación de una base de datos que extraiga datos de plataformas como The StoryGraph, Thesaurus, LastFM y Spotify. Esta BBDD será una parte fundamental del algoritmo. Más aún, será necesaria la creación de una página web, programada con Django y React, para que los usuarios puedan interactuar con el algoritmo.

Este proyecto no podría completarse sin crear unas pruebas de validación tanto de la página web como del algoritmo. Por tanto, las pruebas se dividirán en dos partes: la primera parte se centrará en la usabilidad de la página web, mientras que la segunda parte se centrará en la eficacia del algoritmo. Estas pruebas buscan acreditar y legitimar el algoritmo y, con él, este trabajo.

Palabras clave

Literatura, música, 'Book-to-Soundtrack', sistemas de recomendación, algoritmo, The StoryGraph, Thesaurus, LastFM, Spotify.

Abstract

There is a relationship between literature and music that, although difficult to explain, exists implicitly nonetheless. The aim of this work is to make the implicit explicit through the creation of an algorithm (named 'Book-to-Soundtrack') that receives as input a book and returns music recommendations based on aspects of the book and its emotionality. It is a project adjacent to recommendation systems, and it takes knowledge from existing literature on the subject, though it is not quite the same.

For the adequate attainment of the algorithm, a database that extracts data from platforms such as The StoryGraph, Thesaurus, LastFM, and Spotify will be necessary. This DB will be a fundamental part of the algorithm. Furthermore, the creation of a website will be needed, coded with Django and React, so that the users may interact with the algorithm.

This project could not be complete without testing both the website and the algorithm. Testing, thus, will be divided into two parts: the first part will focus on the website's usability, and the second part will focus on the efficacy of the algorithm. These tests have the aim of crediting and validating the algorithm and, with it, this project.

Keywords

Literature, music, 'Book-to-Soundtrack', recommendation systems, algorithm, The StoryGraph, Thesaurus, LastFM, Spotify.

Índice

1 Introducción **1**

2 Estado del arte **3**

2.1 Sistemas de recomendación 3

2.2 Taxonomías y folksonomías 4

2.3 Sistemas de recomendaciones 'cross-domain' 6

2.4 La taxonomía de The StoryGraph 7

3 Diseño **9**

3.1 Estructura del proyecto 9

3.2 Análisis de requisitos 10

3.2.1 Requisitos funcionales 10

3.2.2 Requisitos no funcionales 12

3.3 Modo en línea versus modo fuera de línea 13

3.4 Diseño 13

3.4.1 Algoritmo 'Book-to-Soundtrack' 15

3.4.2 Base de datos 15

3.4.3 Página web 16

3.5 Diagrama de Gantt 18

4 Desarrollo **19**

4.1 Algoritmo 'Book-to-Soundtrack' y base de datos 19

4.1.1 Libros: The StoryGraph 21

4.1.2 Sinónimos: Thesaurus 22

4.1.3 Música: LastFM 23

4.1.4 Música: Spotify 24

4.2 Página web 25

4.2.1 Back-end 25

4.2.2 API REST 26

4.2.3 Front-end 27

4.2.4 Despliegue en producción 28

5 Pruebas **29**

5.1 Primer cuestionario: usabilidad de la página web 29

5.2 Segundo cuestionario: lista de reproducción de música 34

6 Conclusiones y trabajo futuro 39

Bibliografía 41

Apéndices 43

A Scrapers 45

B Página web 53

Listas

Lista de algoritmos

4.1 Pseudocódigo del algoritmo 'Book-to-Soundtrack'. 20

Lista de códigos

A.1 Código correspondiente de la función `book_search` de **storygraph_scraper.py**. . . . 45
codes/python/scrapers/storygraph_scraper.py 46
A.2 Código correspondiente a la función `book_info` de **storygraph_scraper.py**. 46
codes/python/scrapers/storygraph_scraper.py 47
codes/python/scrapers/storygraph_scraper.py 48
A.3 Código de la función `thesaurus` del fichero **dictionary_scraper.py**. 49
A.4 Código de la función `get_lastfm_music` del fichero **lastfm_scraper.py** 50
codes/python/scrapers/lastfm_scraper.py 51
A.5 Código de la función `spotify_search` del fichero **spotify_filter.py**. 52

B.1 Código del fichero **settings.py**. 53
codes/python/web/settings.py 54
codes/python/web/settings.py 55
B.2 Código del fichero **urls.py**. 56
B.3 Código del fichero **models.py**. 56
codes/python/web/models.py 57
codes/python/web/models.py 58
B.4 Código del fichero **get_track_pool.py**. 58
codes/python/web/get_track_pool.py 59
B.5 Código del fichero **views.py**. 59
codes/python/web/views.py 60
codes/python/web/views.py 61
B.6 Código del fichero **App.jsx**. 61
B.7 Fichero **requirements.txt**. 62
B.8 Fichero **build_production.sh**. 62
B.9 Fichero **package.json**. 63

Lista de figuras

2.1 Capturas de pantalla de The StoryGraph 7
2.2 Conjunto de sinónimos de Thesaurus para la etiqueta ‘dark’. 8

3.1 Diagrama de la organización del proyecto por módulos. 9
3.2 Diagrama de flujo de un usuario interactuando con el algoritmo ‘Book-to-Soundtrack’ . 14
3.3 Diagrama Entidad-Relación de la BBDD 16
3.4 Diagrama de Secuencia del modo en línea de la página web 17
3.5 Diagrama de Gantt de la implementación del proyecto 18

4.1 Diagrama definitivo de la implementación de la BBDD. 20
4.2 Objeto JSON que almacena la información definitiva de un libro de The StoryGraph. . . 22
4.3 Consultas a la BBDD sobre sinónimos de Thesaurus. 22
4.4 Consulta con el número de canciones de LastFM agrupadas por las tres vías para obtenerlas. 24
4.5 Consultas a la BBDD sobre canciones de Spotify. 25
4.6 Diferenciación entre el modo en desarrollo y en producción usado en **settings.py**. . . . 28

5.1 Preguntas sobre edad y género de los participantes 30
5.2 Distribución de la letra elegida para la búsqueda de un libro fuera de línea. 30
5.3 Preguntas sobre el modo fuera de línea de la búsqueda de un libro. 31
5.4 Preguntas sobre el modo en línea de la búsqueda de un libro. 31
5.5 Preguntas sobre la navegación por la página web. 32
5.6 Pregunta ‘Puntúa, del 1 al 5, cuánto problema han supuesto los tiempos de espera/refrescar la página’. 32
5.7 Preguntas sobre la experiencia general con la página web. 33
5.8 Preguntas sobre hábitos de lectura y escucha de música. 34
5.9 Respuestas a la pregunta ‘En los momentos que estás leyendo, ¿sueles escuchar música de acompañamiento?’. 35
5.10 Respuestas a la pregunta ‘¿Te ha parecido una propuesta interesante/curiosa?’. 35
5.11 Preguntas sobre la lista de reproducción. 36
5.12 Preguntas sobre la relación entre lista de reproducción y el libro. 37

Lista de tablas

4.1 Tabla en la que se explican los distintos controladores en **views.py**. 26
4.2 Tabla que muestra las distintas rutas de la página web. 27

5.1 Tabla con la distribución de respuestas para la pregunta ‘Puntúa la página web sobre 10’. 33
5.2 Tabla con la distribución de respuestas para la pregunta ‘Puntúa, del 1 al 5, cuán acertada es la playlist de música generalmente en relación con tu libro’. 37

1

Introducción

El ocio y los 'hobbies' o intereses son una parte vital de la experiencia humana. Es inevitable que, para vivir una vida digna y equilibrada, una persona ha de descansar tanto como trabaja, aunque vivamos en una era en la que esto, normalmente, no sea el caso. Los intereses que se le ocurren a una persona por una pasión descontrolada, una fijación que ocupa su mente la mayoría del día o, incluso, un antojo caprichoso que dura tres semanas, no son solo un atributo de ella, sino un eco de aquello inefable que le hace ser humano. Son esos intereses profundamente desinteresados en los cuales vive la inventiva, la creatividad, la curiosidad y varias características más que hacen que la vida merezca ser vivida.

Dos de estos intereses, los cuales destacan por tener tradiciones que abarcan siglos, son la literatura y la música. La literatura y, más concretamente, la unidad narrativa y temática a la que llamamos novela, se dice que tiene su origen en Don Quijote de la Mancha, de Miguel de Cervantes. Por supuesto, para la vasta mayoría de la población hoy en día, es una obra completamente inaccesible, pero abrió las puertas a un mundo hasta entonces inexplorado. Pocas cosas alimentan la imaginación y la introspección como una buena novela de ficción, llena de sus giros de trama, sus absorbentes conflictos personales y esa estructura invisible que un buen autor sabe incorporar a su texto para elevarlo al estatus de arte.

La música, por otro lado, ha tenido una existencia omnipresente en la vida del ser humano desde que éste posee capacidad de ritmo. El fenómeno moderno de la música, sin embargo, con sus variados géneros, es un asunto que comenzó apenas hace un siglo. Antes, era la música clásica la única que pervivía y que tenía un impacto en la cultura, siendo el perfil de compositor uno que solo tenía cabida para un perfil extremadamente angosto de persona. Fue gracias a la masificación de los medios y, en parte, al acceso indiscriminado a una educación digna para todo el mundo, que la población tuviera un mayor grado de impacto en la cultura, hecho que influyó en la creación de música. En el siglo XX proliferaron muchos géneros que, por primera vez en la historia, reflejaban la diversidad humana.

Es una de las uniones entre la literatura y la música, el verso y la melodía, el fenómeno de la 'canción moderna': aquella que tiene letra. La gente, al parecer, desea una mezcla entre estos dos artes y, por primera vez, tienen suficiente impacto para que este fenómeno se amplifique y se convierta

en algo casi ubicuo. Es casi imposible pensar en música sin pensar en letra de acompañamiento.

Pero esto no es todo: los artistas han estado, desde siempre, fascinados por obras de la literatura y han compuesto canciones directamente inspiradas en ellas. Ejemplos como la maravillosa 'Wuthering Heights' de la etérea Kate Bush, basada en la novela del romanticismo gótico del mismo nombre, o 'Love Story', el eterno clásico de Taylor Swift inspirado en Romeo y Julieta. Y, sin irnos más lejos que en España, el álbum 'El Mal Querer' de Rosalía, ecléctico y experimental, basado en 'Flamenca', un texto anónimo olvidado del siglo XIII que, desde que se hizo notorio, se considera un clásico feminista. He aquí un ejemplo sobre cómo la música moderna puede reavivar algo tan antiguo como la literatura medieval.

La relación entre estos dos artes también vive en lo mundano, pues se puede encontrar en Internet artículo tras artículo, y numerosos posts de Reddit, de personas preguntando por 'canciones que les podría recordar a su libro favorito', o 'libros que causan un efecto parecido a su música preferida'. Y es aquí donde este proyecto toma vida, pues, ¿y si se pudiera crear una forma mecánica de generar recomendaciones de uno dado el otro? Este es uno de los puntos más fuertes de la informática: la automatización de procesos, y llevar dichos procesos a una escala a la que la mano humana no puede llegar.

Presento, pues, la razón de ser de este Trabajo de Fin de Grado: el algoritmo 'Book-to-Soundtrack'. Una herramienta que, basándose en un libro, sugiere música personalizada para ese libro, que logre desbloquear la emocionalidad común en estos dos artes y, en base a eso, actúe como un traductor.

Es importante mencionar la adyacencia de este proyecto con los sistemas de recomendación, aunque no sea exactamente eso lo que se implementa, pues un sistema de recomendación se incorpora encima de un producto ya desarrollado y tiene inteligencia artificial. Este proyecto es, más bien, el propio producto, pues no hay antecedentes de nada parecido que exista. Tal vez, en un futuro, se pueda diseñar un sistema de recomendación basado en esta herramienta.

Para terminar esta introducción, bien puede ser que este trabajo peque de caprichoso, de frugal o de inocuo, que no tenga viabilidad comercial por la completa falta de provecho que tiene; sin embargo, yo argumentaría que es precisamente eso lo que le da vida.

2

Estado del arte

Este proyecto presenta un problema complejo y novedoso que, a su vez, se divide en varios subproblemas. Por ello, no solo se discutirá el estado actual de los sistemas de recomendaciones (el ámbito tecnológico más cercano al algoritmo que se quiere desarrollar) dirigidos a los dominios de los libros y de la música, sino que también es necesario investigar cómo combinar ambos dominios para generar un sistema personalizado de sugerencias coherente.

2.1. Sistemas de recomendación

Los sistemas de recomendación son una tecnología que, durante la última década, ha experimentado un gran auge. Tienen como objetivo sustituir o aproximar el proceso social de recomendación: el acto de recurrir a expertos o a conocidos para tomar la decisión de adquirir y consumir un producto sin tener suficiente información sobre él.

No es una coincidencia que este auge coincida con el de la inteligencia artificial, pues ésta es útil para, en función de datos existentes (datos de experiencia de usuario), predecir y asignar nuevos datos a usuarios de antemano: una recomendación, en esencia.

En la literatura existen muchos tipos de sistemas de recomendación, tanto en el dominio de los libros como en el de la música, que aprovechan diversas metodologías y datos. Por ejemplo, en la tesis doctoral presentada en [1] se diseña un sistema de recomendación de libros para la Biblioteca de la Facultad de Ingeniería de Sistemas e Informática de la Universidad Nacional Mayor de San Marcos, ubicada en Lima, Perú. Dicho sistema está basado en una ontología asociada a Tesauros, y tiene como propósito ayudar a los usuarios a navegar el gran número de libros que posee la biblioteca para acelerar el proceso de encontrar lo que buscan.

Otro artículo, [2], trata sobre un sistema de recomendación, esta vez sobre recursos educativos, en el que se incorpora el análisis de sentimiento y la computación afectiva para determinar las necesidades del usuario. La forma en la que consigue esto es realizando un análisis facial del usuario a partir de imágenes extraídas de su cámara. En efecto, el análisis de sentimiento es importante, como cabe esperarse, para predecir la necesidad o querencia de un usuario [3]. Como se verá más adelan-

te, esta introducción de las emociones y el estado de ánimo será una parte importante del algoritmo desarrollado.

Si pasamos ahora a revisar los sistemas de recomendación en el dominio de la música, encontramos en [4] un estudio que intenta sugerir música a usuarios basándose en la técnica del filtrado colaborativo, es decir, en base a lo que les ha gustado a otros usuarios con perfiles similares. Además, en otro estudio, [5], se crea un sistema de recomendación basado en un sistema de etiquetado social. Se propone que dicho etiquetado social provenga de redes sociales como Twitter, pues "[reúne] gustos, opiniones y subjetividad de una gran cantidad de usuarios de todo el mundo". Además, sostiene que a través del procesamiento del lenguaje natural en redes sociales como Twitter, se pueden extraer características que identifican los gustos musicales de cada usuario [5]. El uso de este sistema de etiquetado que, por definición, es más laxo y libre, es conocimiento valioso que se aplicará en este trabajo.

Es importante destacar la utilidad que presentan estos sistemas y, por supuesto, el valor económico que le puede suponer a una empresa implementarlos en sus plataformas. Sin embargo, se presenta un obstáculo para este trabajo: no se han encontrado precedentes de sistemas de recomendación que sugieran música a partir de un libro (al menos usando Google Scholar). Si bien existen algoritmos especializados en 'cruzar dominios' (del inglés, *cross domain*) [6], se basan en aprovechar patrones globales de interacciones. Aprovechar al máximo las características intrínsecas de un libro concreto para obtener recomendaciones musicales no se ha abordado en el área. Sabiendo esto, se necesitará una investigación más específica sobre el problema para poder, al menos, estrechar la solución.

2.2. Taxonomías y folksonomías

Como ya hemos comentado, los sistemas de etiquetado de un dominio son útiles para sistemas de recomendación y personalización. Al fin y al cabo, ser capaces de describir, por ejemplo, el dominio de la música utilizando un 'conjunto de palabras' acertado y completo lleva naturalmente a una recomendación, a su vez, acertada y completa.

Un concepto en la literatura existente resalta y nos proporciona una forma de manejar el problema; el de una '**folksonomía**'. Tradicionalmente, se utiliza una taxonomía como sistema de etiquetado en un dominio: es un sistema estático e invariante, normalmente creado por un pequeño panel de expertos en el dominio relevante y que tiene como objetivo cubrir el máximo número de facetas en ese dominio. Hacer que un usuario interactúe con esta taxonomía resultaría, como cabe esperar, en unos buenos datos para realizar un sistema de recomendación. Sin embargo, es importante recalcar que las taxonomías de la música, normalmente, son deficientes [7].

Existen varios problemas al intentar crear una taxonomía definitiva para la música. Para empezar, se encuentra que la industria musical usa taxonomías inconsistentes. Además, una taxonomía de

música definida por expertos presenta dos problemas principales. El primero es que suelen ser incompletas, ya que es imposible que un pequeño grupo de expertos incorpore en una única estructura todo lo que es relevante para el dominio de la música. El segundo es que una taxonomía puede quedar fácilmente anticuada: constantemente, en el mundo de la música surgen nuevos géneros y técnicas, lo que hace que una clasificación estática no sea ideal [7].

Alternativamente, plataformas de música como LastFM presentan un sistema de clasificación y descripción de sus elementos (artistas, álbumes o canciones) que recientemente se ha denominado 'folksonomía'. Una *folksonomía* se define como un sistema de etiquetado social, donde un usuario puede asignar a un elemento una etiqueta elegida libremente. El efecto que tiene esto es la creación de un sistema de clasificación no estructurado [8]. Se diferencia de una taxonomía, pues esta es un sistema rígido, normalmente predefinido por el sistema o por expertos en el tema, y que es limitante en su escala por diseño.

En efecto, una folksonomía evita estos problemas; por su expansividad y variabilidad, son bastante más aptas para cubrir todas las 'formas de hablar sobre música', y también están siendo continuamente actualizadas por los usuarios.

Además, existe un efecto que suele aparecer en una folksonomía de cualquier dominio. Primero, exploremos este efecto en un dominio de música; de nuevo, usaremos LastFM. Un análisis generalizado de las etiquetas usadas muestra una distribución 'altamente desequilibrada': las 200 etiquetas más utilizadas componen un 50 % de las instancias, mientras que un porcentaje significativo de las etiquetas aparece solo una vez [9], lo que se conoce como el *efecto cola*.

Dicho efecto es muy común en folksonomías de cualquier dominio: es muy probable que los usuarios usen etiquetas altamente personales y que, por tanto, solo sean relevantes para dicho usuario, lo cual genera una gran cantidad de etiquetas 'individualizadas' que acaban constituyendo una proporción importante del sistema.

Sin embargo, esto muestra verdaderamente el poder de una folksonomía, pues, incluso sin la estructura externa de una taxonomía, existe un pequeño porcentaje de etiquetas que dominan las clasificaciones; parece ser que los usuarios, por su cuenta, acaban haciendo una clasificación coherente de cada dominio. Por ejemplo, el artista Elton John, en LastFM, es clasificado con las etiquetas "70s", "80s", "pop", "classic-rock", "singer-songwriter" y "british", entre otras [7]. Lo que podría llevar a una clasificación caótica y errática acaba encontrando su propio orden.

Lo que es todavía más importante: existe en LastFM una clara distribución de etiquetas por categoría. 'Género', 'opinión', 'estado de ánimo' e 'instrumento' son las cuatro categorías más usadas, y representan un 76,3 % del total [9].

Conviene detenerse en la categoría de 'estado de ánimo' ('mood' en inglés), responsable de un 14,4 % de las etiquetas según [9] y que, como su nombre indica, representa el estado de ánimo de

la canción. Este subconjunto de etiquetas, como puede esperarse, también presenta una distribución desequilibrada, con un minúsculo porcentaje de las etiquetas más usadas siendo altamente sobrerrepresentadas en el sistema. De nuevo, esta clasificación es posible por la forma en la que funciona una folksonomía.

Cabe destacar la inherente subjetividad de asociar un estado de ánimo a una canción. Sin embargo, los usuarios asocian la música que escuchan con estados de ánimo con relativa frecuencia en LastFM. Esta ambigüedad es una característica definitoria de las folksonomías. Aquí se encuentra un potencial inmenso para construir un sistema personalizado de sugerencias, y es la clave para el sistema que se pretende construir en este proyecto.

En la siguiente sección se usará el concepto de la folksonomía para abordar el problema del cruce de dominios.

2.3. Sistemas de recomendaciones 'cross-domain'

Existe literatura que aborda la asociación de folksonomías de distintos dominios (llamadas 'cross-domain' o de dominios cruzados). El objetivo es sencillo: traducir el lenguaje de un dominio al otro, con el propósito de que un sistema de recomendación pueda cruzarse entre dominios. Si a un usuario le gusta un elemento de un dominio, le podría gustar este otro elemento de otro dominio.

Ya se ha intentado asociar la folksonomía de la música con la del cine. En [8], se asocia cada folksonomía, altamente específica en su dominio, con una folksonomía de emociones independiente del dominio, y de esta forma se realiza una 'traducción' del lenguaje del cine al lenguaje de la música. Esto muestra una realidad importante: el estado de ánimo y la emocionalidad son el puente mediante el cual se pueden (y tal vez se deban) basar sistemas de recomendaciones de dominios cruzados. Es esa filosofía la que ha inspirado este proyecto.

El problema que queda por abordar, entonces, es realizar el cruce entre el dominio de los libros y el dominio de la música. Esto es más problemático que asociar la música con el cine. Se ha demostrado que el estado de ánimo es una parte innata en la clasificación de la música y surge de forma natural dentro de una folksonomía. Sin embargo, falta analizar cómo se vería una folksonomía de libros y si se puede hacer un cruce.

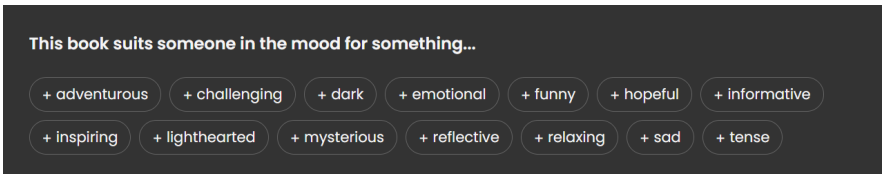
Un ejemplo paradigmático de folksonomía de libros lo encontramos en la plataforma LibraryThing, que tiene un sistema de etiquetado social que genera una folksonomía. Se encuentra que las etiquetas más relevantes que surgen para los usuarios se suelen encontrar en tres categorías, que son 'tema/tópico', 'personal' y 'género'. Es notable la ausencia de estado de ánimo o emotividad, la cual no aparece tampoco en otra categoría menos usada [10]. Tal vez la categoría 'personal' podría contener cierta información asociada al estado del ánimo, pero no es suficiente como para crear su propia categoría.

Esto revela una diferencia fundamental en la forma en que los usuarios describirían una canción con respecto a un libro, obviando categorías como el género: mientras que en la primera se evocan estados de ánimo, en la segunda se prefiere evocar los temas o *topics*, sin interesarse demasiado en cómo un libro les hace sentir directamente.

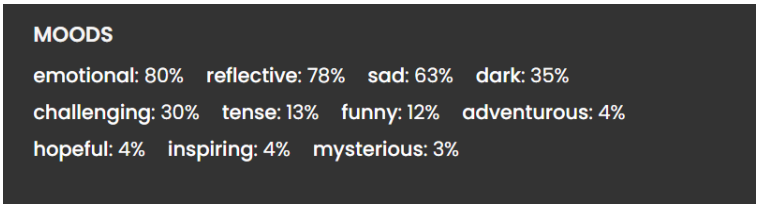
Todo esto, combinado con la falta de literatura al respecto, nos obliga a acortar el alcance del problema. Se usará la filosofía de determinar un 'intermedio de emociones' que pueda unir ambos dominios.

2.4. La taxonomía de The StoryGraph

La solución encontrada (que, por diseño, es imposible que sea la única) proviene de una plataforma de libros llamada The StoryGraph, menos popular que LibraryThing pero que posee una cualidad especial en su descripción y clasificación de libros que la diferencia: en las reseñas de un libro, puedes etiquetarlo en base a una taxonomía de catorce 'moods' o estados de ánimo (véase la Figura 2.1(a)).



(a) Taxonomía de estado del ánimo de The StoryGraph.



(b) Distribución por porcentaje del estado de ánimo de un libro en The StoryGraph

Figura 2.1: Capturas de pantalla de The StoryGraph

Esta taxonomía contiene términos que, en su mayoría, se pueden visualizar como una etiqueta de estado de ánimo más en una folksonomía de música (siendo tal vez la etiqueta 'informative', o informativo, la que menos encaja, lo cual será relevante más adelante) y constituye el punto de unión que se buscaba para plantear un cruce entre ambos dominios.

Una limitación importante de esta solución es, evidentemente, que el dominio de libros se basa en una taxonomía rígida, dependiente de los creadores de The StoryGraph. Sin embargo, como ya vimos anteriormente, una folksonomía de libros no lleva a una categoría diferenciada de estados de ánimo. En ese sentido, esta taxonomía anima a los usuarios a pensar en cómo les ha hecho sentir un libro y en el estado de ánimo que podría llevar una persona a leerlo, que es lo que se buscaba. Otro apunte

que se puede hacer es que esto no constituye un cruce de dominios completo, sino que reduce el problema a cruzar etiquetas que se usan para describir el 'mood' de los libros a la música.

Siguiendo con el análisis de The StoryGraph, en el apartado de descripción de un libro de la plataforma se pueden observar las etiquetas de estado de ánimo ordenadas en función del porcentaje de personas que, en su reseña, decidieron otorgar a dicho libro esa etiqueta.

Esto introduce un nivel adicional de variabilidad, ya que genera una proporción entre las etiquetas en vez de considerarlas por igual, lo cual contribuye a una distribución única de estado de ánimo para cada libro.

Este trabajo tiene, en definitiva, el objetivo de usar la taxonomía de The StoryGraph para obtener la 'composición emocional' de un libro. Se usará LastFM y su folksonomía (y, específicamente, la subcategoría de estado de ánimo) para hacer el cruce de dominios y obtener así un sistema de recomendación altamente novedoso hasta donde hemos explorado.

Existe el problema de reducir el amplio dominio de los libros a catorce términos, una clasificación sin duda incompleta y limitante. En el caso que no se intentara hacer una expansión de esta idea, en LastFM solamente se harían búsquedas de música basadas en esas catorce etiquetas, lo cual generaría, como máximo catorce 'categorías' de canciones.

Por este motivo, se ha decidido que, para cada término de la taxonomía, se realizará una búsqueda de sinónimos, y serán los sinónimos los que se busquen como etiquetas en LastFM. La estructura resultante serán catorce meta-categorías que se descomponen en un número indeterminado pero evidentemente más grande de categorías de música. Específicamente, se usará la plataforma Thesaurus como recurso para obtener los sinónimos de cada término. La Figura 2.2 muestra un ejemplo de sinónimos asociados con la etiqueta 'dark'.

Figura 2.2: Conjunto de sinónimos de Thesaurus para la etiqueta 'dark'.

3

Diseño

Se comienza este capítulo explicando la estructura del proyecto, dividiéndolo en módulos funcionales que interactúan entre ellos para crear un sistema cohesivo. A continuación, se discute el análisis de requisitos. Tras una sección explicando dos modos diferenciados del algoritmo, se muestra el diseño de cada uno de los módulos del proyecto. Se termina con un diagrama de Gantt detallando la organización temporal del proyecto y sus módulos.

3.1. Estructura del proyecto

La Figura 3.1 muestra la distribución del proyecto por módulos. Cada módulo existe con cierta independencia, pero los módulos interactúan entre sí de tal forma que generan una estructura cohesiva y fluida.

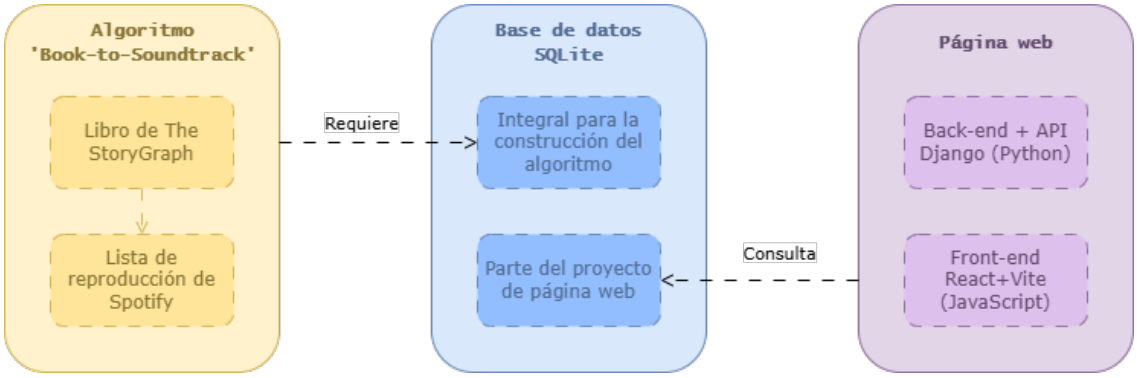


Figura 3.1: Diagrama de la organización del proyecto por módulos.

El módulo de color amarillo es el algoritmo 'Book-to-Soundtrack'. Antes que nada, es importante explicar que, debido a que LastFM no está optimizado para ofrecer enlaces de reproducción de la música que tiene, se ha decidido usar Spotify como plataforma de streaming. Las canciones obtenidas a partir de LastFM se buscarán en Spotify para incorporar a cada una su enlace de reproducción. Por tanto, el algoritmo tiene como parámetro de entrada un libro de The StoryGraph y devuelve como salida una lista de reproducción de Spotify. Se considera el propósito inicial del proyecto, pero el resto de los

módulos lo amplían y complementan.

El módulo de color lila corresponde a un proyecto full-stack de página web. El propósito de este módulo es, principalmente, presentar el algoritmo de tal forma que un usuario pueda interactuar con él y luego pueda participar en el estudio posteriormente. Adicionalmente, se ha pensado que el back-end y el front-end sean dos aplicaciones separadas y conectadas por una API. El objetivo de esto es crear una interfaz de usuario dinámica y fluida, por lo que se vio necesario usar un framework especializado en front-end.

El módulo de color azul representa la base de datos del proyecto y es el módulo conectivo. Evidentemente, la BBDD forma parte del proyecto de la página web, específicamente del back-end, y es imprescindible para su funcionamiento. Por otro lado, la BBDD, como quedará claro en el resto de la memoria, es una parte fundamental del algoritmo, ya que, a medida que el flujo del algoritmo se va construyendo, también se llena la BBDD. Además, el diseño y desarrollo de la BBDD refleja el diseño y desarrollo del propio algoritmo.

3.2. Análisis de requisitos

En este módulo se presentan los requisitos del proyecto, divididos por los módulos presentados en la sección anterior. A su vez, los requisitos se dividen en requisitos funcionales y requisitos no funcionales. Los requisitos funcionales detallan las capacidades y operaciones específicas que el sistema es capaz de realizar. Los requisitos no funcionales tratan sobre la escalabilidad, la seguridad y la fiabilidad del sistema, entre otros, con el propósito de ofrecer una experiencia de usuario robusta.

3.2.1. Requisitos funcionales

Algoritmo 'Book-to-Soundtrack'

RF-ABS-1.- El algoritmo 'Book-to-Soundtrack' estará compuesto por tres submódulos: la búsqueda de libros en The StoryGraph, la expansión por sinónimos en Thesaurus y la búsqueda y entrega de una lista de reproducción de música.
RF-ABS-2.- El algoritmo accederá exitosamente a la información de un libro en la plataforma de libros The StoryGraph.

RF-ABS-2.1.- El algoritmo recibirá, como parámetro de entrada, una cadena de texto que contenga información identificatoria del libro, como el ISBN, por ejemplo.

RF-ABS-2.2.- El algoritmo devolverá en su salida la información característica del libro.

RF-ABS-2.3.- Dicha información incluirá las etiquetas de estado de ánimo que caracterizan a ese libro, así como sus porcentajes de afinidad con cada etiqueta.
RF-ABS-3.- El algoritmo recopilará, a partir de las etiquetas de estado de ánimo, la lista de sinónimos

correspondientes a cada una de la plataforma Thesaurus.

RF-ABS-4.- El algoritmo procesará la lista de sinónimos y devolverá una lista de reproducción de música acorde.

RF-ABS-4.1.- La lista de reproducción tendrá una distribución de canciones similar a la distribución de afinidad de las etiquetas del libro.

RF-ABS-4.2.- El algoritmo pedirá como parámetro el número de canciones de la lista.

RF-ABS-4.3- El algoritmo realizará primero una búsqueda de todas las canciones guardadas que estén asociadas a alguna etiqueta de la lista.

RF-ABS-4.4.- El algoritmo, a partir de la lista completa de canciones, obtendrá al azar un subconjunto de canciones, cuyo número coincidirá con el anteriormente especificado, y cuya proporción coincidirá con la proporción de las etiquetas de estado de ánimo.

RF-BD-5.- Se programarán dos modos del submódulo de libros del algoritmo: el modo en línea y el modo fuera de línea.

RF-BD-5.1- El modo en línea obtiene la información del libro directamente de The StoryGraph, usando tecnología como una API o web scraping.

RF-BD-5.2.- El modo fuera de línea obtiene la información de la BBDD.

Base de datos

RF-BD-1.- La BBDD estará compuesta por cinco entidades: Libro de The StoryGraph, Etiqueta de The StoryGraph, Sinónimo de Thesaurus, Canción de LastFM y Canción de Spotify.

RF-BD-1.1.- Cada una de estas entidades estarán unidas entre ellas, en el orden en el que se mencionan, por relaciones de muchos a muchos.

RF-BD-2.- El proceso de poblar la BBDD ocurrirá en paralelo con la construcción del algoritmo.

RF-BD-2.1.- Se recopilará una cantidad de entre 500 y 1000 libros de The StoryGraph con su respectiva información relevante, incluidas su unión con las etiquetas correspondientes.

RF-BD-2.2.- Se recopilarán las 14 etiquetas de estado de ánimo que conforman la taxonomía de The StoryGraph. En la tabla de relación de muchos a muchos entre Libro y Etiqueta, habrá un atributo que indicará el porcentaje de afinidad de un Libro con una Etiqueta.

RF-BD-2.3.- Se recopilarán los sinónimos de Thesaurus asociados a cada una de las 14 etiquetas, y se enlazarán de forma correspondiente en la BBDD. Se incluirán las 14 etiquetas dentro de la tabla de Sinónimos.

RF-BD-2.4.- Se recopilarán las canciones de LastFM más relevantes para cada palabra en la lista de sinónimos.

RF-BD-2.5.- Para cada canción recopilada, se hará una búsqueda en Spotify y se obtendrá su enlace de reproducción, para luego insertarlo en la tabla correspondiente.

RF-BD-3.- Toda búsqueda fuera de línea que haga el algoritmo serán consultas a la BBDD.

Página web

RF-WEB-1.- La página estará compuesta por dos aplicaciones: el back-end y el front-end. Se hará una API que comunique ambas aplicaciones.
RF-WEB-2.- El algoritmo 'Book-to-Soundtrack' se adecuará a la estructura de la página web, y todos sus componentes trabajarán como uno para implementarlo.

RF-WEB-2.1.- El front-end proyectará la interfaz de la página web con la que interactuarán los usuarios.

RF-WEB-2.2.- El front-end hará una llamada a la API para pedir información.

RF-WEB-2.3- El back-end recibirá la llamada y, bien sea una consulta a la BBDD o cualquier otra solicitud, devolverá la información solicitada.

RF-WEB-2.4- El front-end recibirá una respuesta de la API con la información solicitada.

3.2.2. Requisitos no funcionales

RNF-1.- Se usará la plataforma de libros The StoryGraph para extraer libros y su información.
RNF-2.- Se usará la plataforma Thesaurus para recabar sinónimos de términos.
RNF-3.- Se usará LastFM como plataforma de música, y se usará la biblioteca de Python `pylast` para extraer música de la misma.
RNF-4.- Se usará la API de Spotify para complementar la información musical proporcionada por LastFM.
RNF-5.- La suma total de procesos que conforman el algoritmo no superarán los 10 segundos de duración.
RNF-6.- La BBDD se hará en un fichero de SQLite.
RNF-7.- Cualquier consulta a la BBDD no tardará más de 3 segundos.
RNF-8.- Habrá suficiente cantidad y variedad de libros y de música en la BBDD para asegurar una buena experiencia de usuario con el algoritmo.
RNF-9.- El back-end y la API se desarrollarán en Django.
RNF-10.- El front-end se desarrollará con React, usando Vite como herramienta para construir el proyecto desde cero.
RNF-11.- El despliegue a producción de ambas aplicaciones se gestionará en Render.
RNF-12.- La página web estará disponible al menos un 95 % del tiempo
RNF-13.- Los tiempos de carga de las llamadas a la API no supondrán más de 15 segundos
RNF-14.- La página web será capaz de cargar con, al menos, 10 usuarios accediendo simultáneamente sin que se vea afectado el rendimiento de la misma.
RNF-15.- La página web será compatible con todos los navegadores y dispositivos modernos.
RNF-16.- La página web dispondrá de una interfaz intuitiva y fácil de navegar, con un diseño dinámico y adaptable.

3.3. Modo en línea versus modo fuera de línea

Debido a que se ha hecho mención de esto en el quinto requisito funcional del algoritmo, se considera importante discutir las implicaciones de diseño que tendría crear dos modalidades distintas del algoritmo: un modo en línea y otro fuera de línea.

El modo en línea, como su nombre indica, implica acceder a los servicios web (páginas web, APIs, etc.) para extraer la información relevante en el momento de procesar la petición del algoritmo. Sus puntos a favor son: la información extraída estará actualizada, y se podrá acceder a cualquier recurso que se quiera, de forma ilimitada. Sus puntos en contra son: el tiempo de espera elevado que supone, así como una probabilidad no nula de que falle alguna petición de información en un punto intermedio.

El modo fuera de línea implica acceder a una base de datos para extraer la información pertinente en cada paso intermedio del algoritmo. Un requisito necesario para este modo es, en efecto, el diseño e implementación de una base de datos local. Sus puntos a favor son: el reducido tiempo de espera que conlleva la extracción de información, así como la probabilidad prácticamente nula de que falle la petición (puesto que se reduciría al acceder a la BBDD). Sus puntos en contra son: se reduce bastante el alcance del algoritmo, debido a que una BBDD produce resultados inherentemente limitados, y la información que se almacena podría variar considerablemente en comparación con su fuente.

En este trabajo, esta dicotomía se aplicará, sobre todo, en el caso de la búsqueda de un libro. Ambos modos poseen ventajas e inconvenientes y, además, la implementación es plausible, por lo que se implementan ambos. De esta forma, se combina la fiabilidad del modo fuera de línea con el mayor alcance del modo en línea.

No es necesario aplicar el modo en línea con los sinónimos, puesto que, al fin y al cabo, hay una cantidad finita de ellos. Cargarlos en la BBDD no hace que se pierda información importante.

Con la música, se estima que, para lo que se quiere hacer, el coste de recuperar la información en línea cada vez que se hace una petición es demasiado alto. Se determina, por tanto, que cargar la música se hará estrictamente fuera de línea. Por tanto, se convierte en una prioridad importante recopilar la mayor cantidad de música posible para asegurar un algoritmo de suficiente calidad.

3.4. Diseño

En esta sección se detalla el diseño de cada uno de los módulos del proyecto por separado y se muestra cómo se unen para crear un proyecto cohesivo.

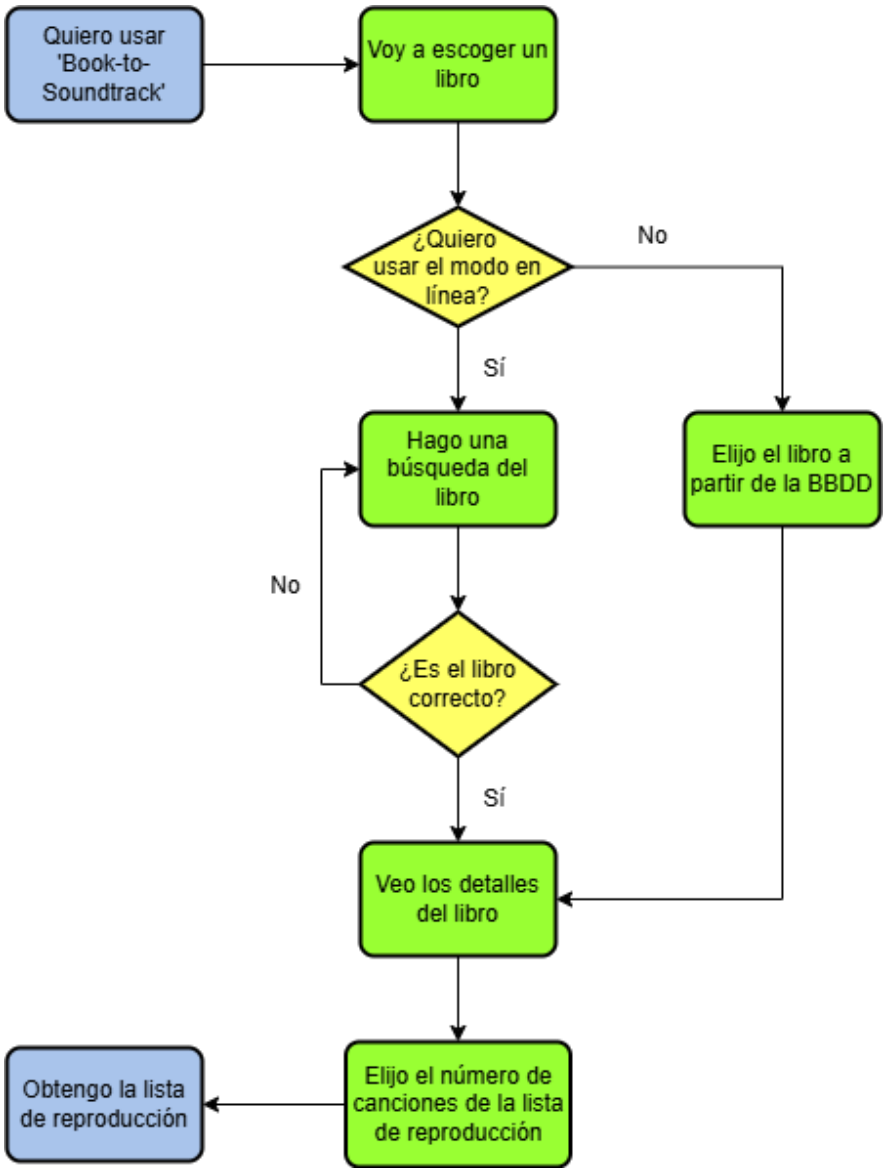


Figura 3.2: Diagrama de flujo de un usuario interactuando con el algoritmo 'Book-to-Soundtrack'

3.4.1. Algoritmo 'Book-to-Soundtrack'

La Figura 3.2 es un diagrama de flujo que representa las interacciones que el algoritmo tiene con un usuario de principio a fin. La entrada es 'Quiero usar el algoritmo' y la salida es 'Obtengo la lista de reproducción'. El primer paso intermedio, y también el más programáticamente complejo, es el de la búsqueda de un libro. Esto es porque, como se estableció en la sección anterior, será el paso que se divida en modo en línea y modo fuera de línea. El sistema comienza haciendo esa pregunta, y el usuario responde. Si elige el modo en línea, realizará una búsqueda del libro y repetirá el paso hasta encontrar el libro buscado. Si elige el modo fuera de línea, elegirá el libro a partir de la lista de libros disponible en la BBDD.

Los siguientes pasos, en orden, son: mirar los detalles del libro, lo cual incluye la composición emocional del libro y, para cada etiqueta de estado de ánimo de esa composición, la lista de sinónimos de la misma; y elegir el número de canciones que tendrá la lista de reproducción. No resultan complejos porque ya no hay divergencias en las interacciones posibles con el usuario.

Otra forma de describir el algoritmo es en función de los submódulos mencionados en sus requisitos funcionales. Los tres submódulos del algoritmo son: módulo de libros de The StoryGraph, módulo de sinónimos de Thesaurus y módulo de lista de reproducción de canciones, el cual, a su vez, puede subdividirse entre LastFM y Spotify. Cada módulo interactúa formando una cadena en el orden que se han presentado y, como se puede esperar, transforma un libro en una lista de reproducción de música.

3.4.2. Base de datos

La base de datos es una parte vitalmente importante de este proyecto, ya que es la herramienta principal para implementar el modo fuera de línea. La Figura 3.3 corresponde al diagrama Entidad-Relación de la BBDD. Puede dividirse aproximadamente en cuatro partes que están interconectadas.

La primera parte corresponde a los libros extraídos de The StoryGraph. La segunda parte corresponde a las etiquetas de The StoryGraph, las que constituyen la taxonomía mencionada en el estado del arte. A pesar de estar unidas como información a cada libro que describen, son una pieza clave del algoritmo y, por tanto, se ha decidido que conformen su propia tabla. La relación entre libro y etiqueta es de muchos a muchos y tiene un atributo llamado 'peso', el cual corresponde al porcentaje de coincidencia con la etiqueta que posee el libro específicamente.

Son las etiquetas de estado de ánimo las que combinan con la tercera parte de esta BBDD: los sinónimos de Thesaurus. Destaca en su relación de muchos a muchos el atributo 'afinidad', el cual hace referencia a la correlación del sinónimo con su palabra, y que puede tener los valores de 'débil', 'fuerte' o 'el más fuerte'. En efecto, es un paralelismo directo a la forma que Thesaurus clasifica sus sinónimos. Es importante mencionar que se incluyen las propias palabras de la taxonomía de The

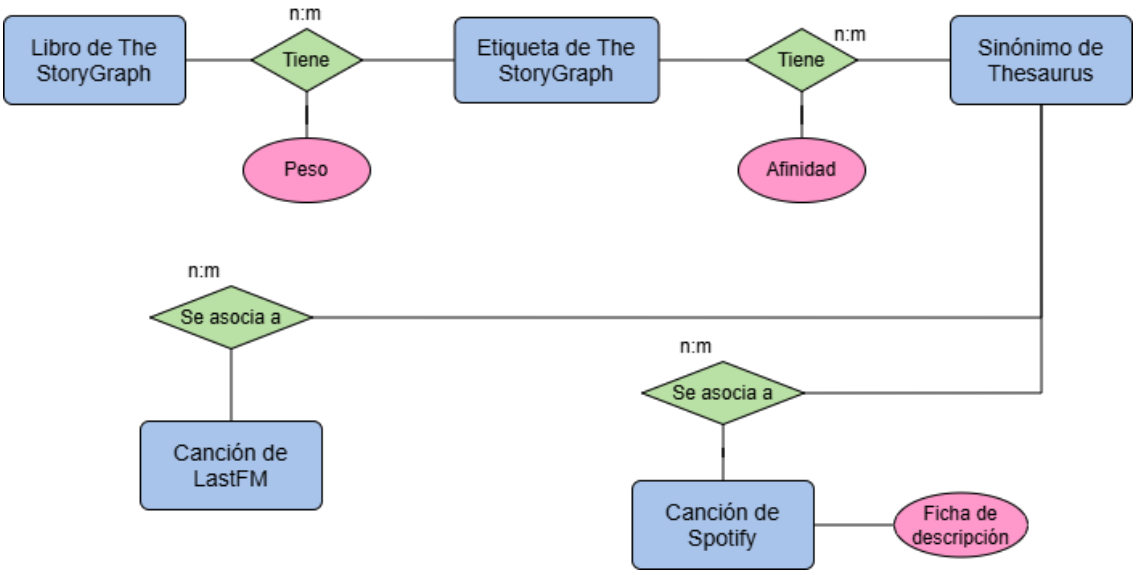


Figura 3.3: Diagrama Entidad-Relación de la BBDD

StoryGraph en el grupo de sinónimos, para que así puedan ser usadas en la búsqueda de música.

La última parte se corresponde a las canciones, y se dividen en dos: las obtenidas por LastFM y las obtenidas por Spotify. Es importante denotar que, pragmáticamente, solo las canciones de Spotify serán usadas en la aplicación del algoritmo final; esto es porque las canciones de LastFM solo tienen un nombre y un artista. Es al pasar el elemento a Spotify cuando se consigue la información requerida (se corresponde al atributo 'ficha de descripción', el cual se almacenará en la BBDD como un objeto JSON) como puede ser el enlace externo de Spotify o una cubierta.

3.4.3. Página web

La estructura de dicha página web consistirá en un servicio front-end, un servicio back-end, y una API que tendrá como propósito intercambiar información entre los servicios.

La Figura 3.4 muestra una interacción exitosa en el modo en línea de la página web. La secuencia de acciones comienza pidiendo un libro a la API, el cual se busca en The StoryGraph y se devuelve. Lo siguiente es mostrar una página de detalles del libro, y se obtiene haciendo una llamada parecida pero mostrando más información, entre la que cabe destacar las etiquetas del libro.

Lo siguiente es mostrar los sinónimos de cada etiqueta, con el propósito de mostrar al usuario los términos que se usarán para la búsqueda de canciones. Por último, se hace una llamada a la API para buscar las canciones en la base de datos; el parámetro 'n' hace referencia al número de canciones que el usuario ha escogido para la lista de reproducción, el cual será entre 5 y 30 canciones, incluidas.

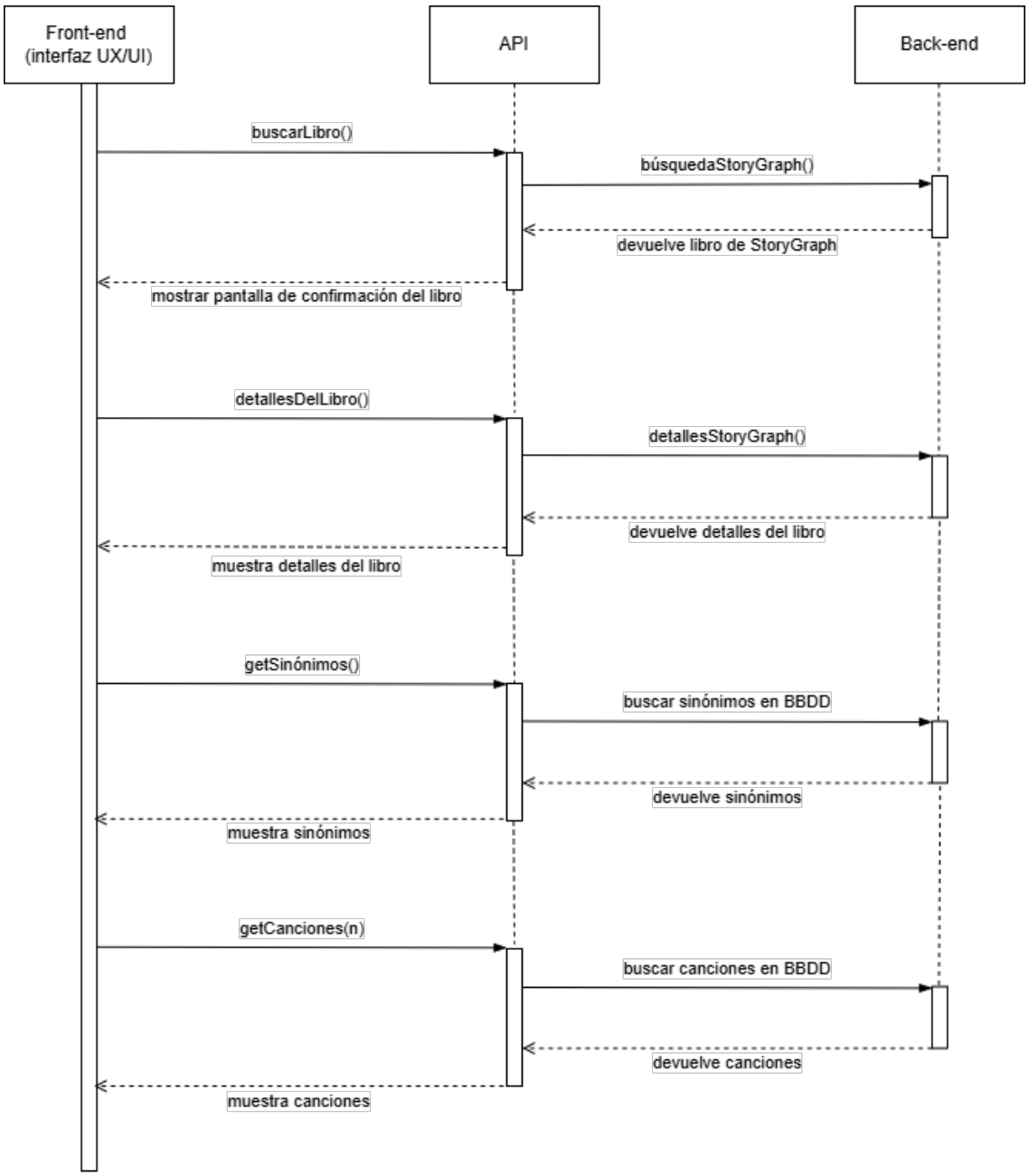


Figura 3.4: Diagrama de Secuencia del modo en línea de la página web

3.5. Diagrama de Gantt

Esta sección muestra y explica el Diagrama de Gantt (véase la Figura 3.5), que tiene como objetivo detallar el flujo temporal del proyecto. Muestra visualmente cómo se estima el comienzo y el fin de cada módulo del proyecto y, en el proceso, presenta un aspecto distinto de cómo interactúan entre sí.

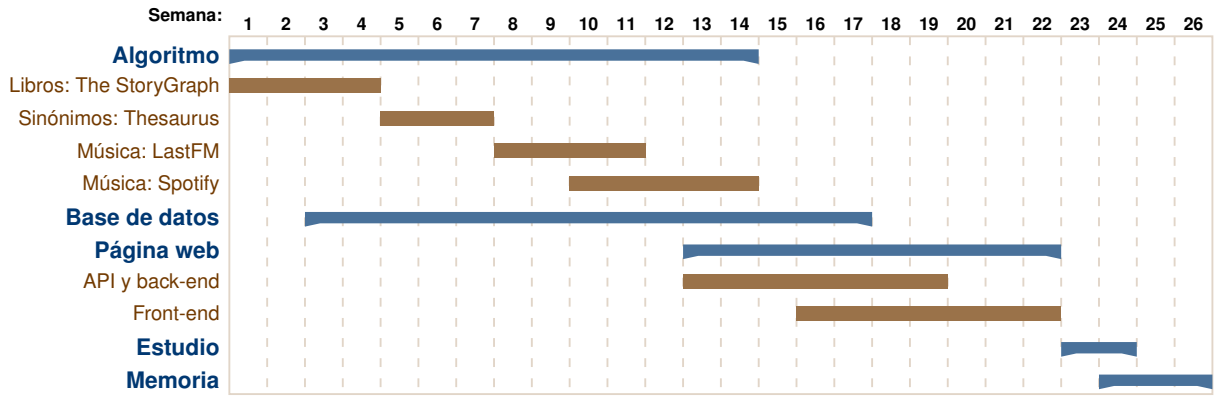


Figura 3.5: Diagrama de Gantt en el que se muestra el desglose del proyecto

Como es de esperar, se comienza con la implementación del algoritmo 'Book-to-Soundtrack'. Además, se puede observar el desglose de esa tarea en sus submódulos. Cabe destacar que la tarea de la base de datos se realiza simultáneamente a partir de la semana 3: esto es porque, como ya se ha mencionado, es una consecuencia natural de crear el algoritmo en cada uno de sus submódulos que se llene la entidad de la BBDD correspondiente.

La siguiente tarea es la página web, la cual tiene también cierta concordancia con las tareas anteriores, pues la BBDD está bastante asociada a la aplicación de back-end. Una vez terminada la página web, se puede realizar el estudio de usuarios. Se termina el proyecto escribiendo la memoria del mismo.

4

Desarrollo

El desarrollo de este proyecto se divide en dos partes: la primera es el algoritmo y la base de datos, módulos que unen por razones que se explican a continuación; la segunda es la página web.

4.1. Algoritmo 'Book-to-Soundtrack' y base de datos

Una decisión necesaria en esta sección es la de explicar a la vez la implementación del algoritmo y de la base de datos. Esto tiene su explicación en el diagrama de Gantt del capítulo anterior, en el cual puede verse un solapamiento temporal muy significativo entre ambas tareas. Sin embargo, el motivo de más importancia es que el propio algoritmo está inextricablemente unido a consultas a la BBDD; no puede existir uno sin el otro. Se muestra el pseudocódigo del mismo en el Algoritmo 4.1. Cabe destacar las cuatro consultas a la BBDD que se hacen en el modo fuera de línea, una más que en el modo en línea.

Una parte importante del pseudocódigo está justo al final: se puede observar cómo el sistema, primero, hace una compilación de todas las canciones asociadas a todos los sinónimos relacionados con todas las etiquetas asociadas al libro. Acto seguido, coge un subconjunto aleatorio del conjunto de canciones y de ahí genera la lista de reproducción. Es importante entender que la elección del subconjunto está determinada por las proporciones de las etiquetas del libro, para mayor personalización.

Para la base de datos se usará un fichero SQLite. Ya se mencionó en el capítulo anterior que la BBDD es fundamental para el funcionamiento del algoritmo, pues, salvo en el modo en línea para obtener un libro, toda la información requerida se conseguirá con una consulta a la susodicha.

En la Figura 4.1 puede verse la base de datos implementada, con una tabla para los libros, las etiquetas, los sinónimos, las canciones de LastFM y las canciones de Spotify. Para representar las relaciones de muchos a muchos, se han creado tablas intermedias con una clave foránea compuesta por las claves primarias de las entidades que se quieren relacionar.

La sección en el repositorio donde se encuentra la investigación y construcción del algoritmo se encuentra dentro de la carpeta **tfg_myproject/project_misc**. Destacan los directorios **scripts**, donde

input : modo de ejecución: enlinea, fueradelinea
output: lista de reproducción

```
1  if modo en linea then
2      while Libro buscado do
3          Leer Libro ;
4          Busca Libro en The StoryGraph ;
5          Escribir Libro de StoryGraph ;
6      end
7  else
8      Escribir BBDD Libros de StoryGraph ;
9      Leer Libro Buscado ;
10     Consulta a BBDD: Libro de StoryGraph ;
11 end
12 Escribir Libro Buscado ;
13 Consulta BBDD: Etiquetas de StoryGraph ;
14 Escribir Etiquetas de Libro ;
15 foreach Etiqueta en Etiquetas de Libro do
16     Consulta BBDD: Sinónimos de Thesaurus ;
17     Escribir Sinónimos de Etiqueta ;
18 end
19 Leer NúmeroCanciones ;
20 foreach Sinónimo en Sinónimos de Etiquetas de Libro do
21     Consulta BBDD: Canciones ;
22     Escribir Canciones de Sinónimo ;
23 end
24 ListaReproducción ← Elegir Aleatoriamente NúmeroCanciones de Canciones ;
```

Algoritmo 4.1: Pseudocódigo del algoritmo 'Book-to-Soundtrack'.

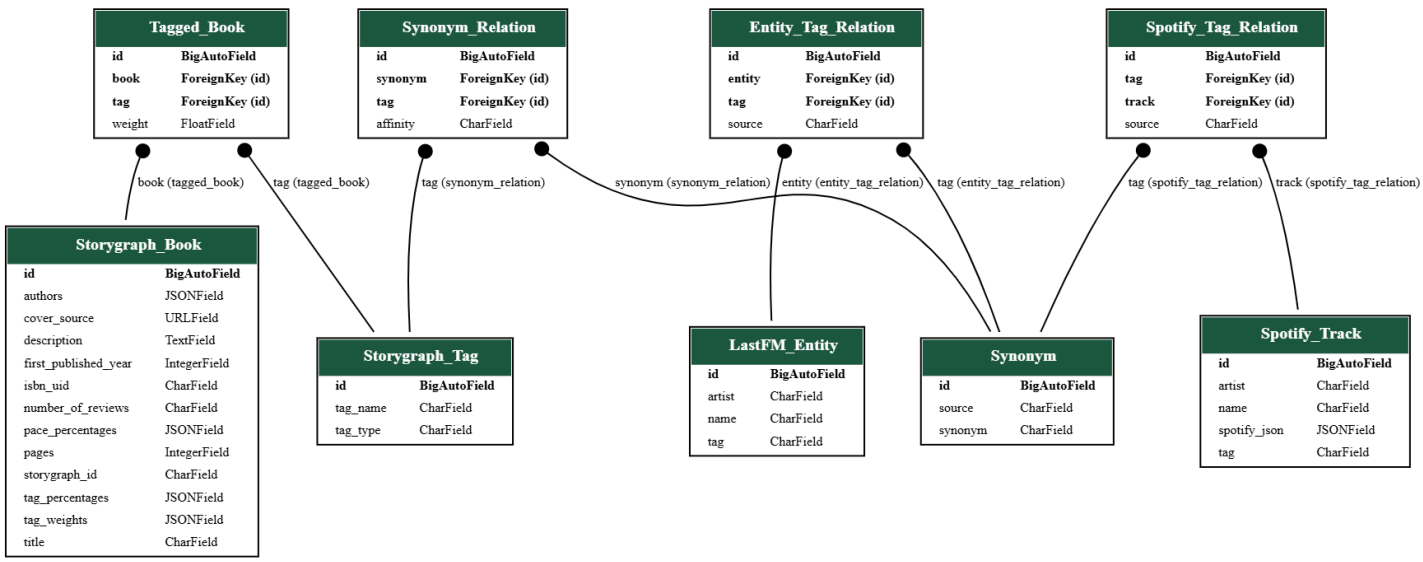


Figura 4.1: Diagrama definitivo de la implementación de la BBDD.

se encuentran todos los ficheros Python, y **json_files**, donde están todos los ficheros JSON.

4.1.1. Libros: The StoryGraph

La primera tarea realizada de todo el desarrollo del proyecto fue encontrar la forma de obtener la información de un libro de The StoryGraph. La primera forma pensada fue usar una biblioteca de Python llamada `storygraph_api`. Destaca el uso de las dos funciones principales de la biblioteca: `book_info` y `search`.

Ambas funciones utilizan tecnología *web scraping* para obtener la información relevante de The StoryGraph. La función `search` necesita como argumento de entrada una consulta de un libro, y devuelve el id específico de The StoryGraph del libro encontrado. La función `book_info` recibe como argumento un id de un libro, y devuelve un objeto JSON con la información del libro.

Usando ambas en consecución permite obtener la información de un único libro a partir de una característica identificatoria del mismo, como puede ser su ISBN o la combinación de título y autores. Esto es muy útil en el modo en línea, en el cual se realiza una única búsqueda; para el relleno de la base de datos para el modo fuera de línea, sin embargo, será necesaria una forma de hacer una búsqueda masiva.

Dicha búsqueda se hará con un programa que importa un fichero JSON, que contiene en cada línea una cadena de texto con título y autor de un libro que se usarán como consultas para conseguir la información de cada libro.

El resultado se guardará en otro fichero JSON, el cual se usará más adelante para rellenar la BBDD. Es importante remarcar el uso de ficheros JSON como receptores intermediarios de la información: debido a que el web scraping y las llamadas a APIs que se realizan tienen una probabilidad no nula de fallar, se ha querido asegurar que, además, no se hace la acción de rellenar la BBDD directamente.

En un momento determinado, por desgracia, la biblioteca `storygraph_api` dejó de funcionar. Evidentemente, es necesaria una alternativa, por lo que se decidió realizar la labor de web scraping manualmente. Destaca la labor de parsear el contenido del fichero HTML obtenido a través del scraping, con el objetivo de conseguir la información requerida.

El fichero **scrapers/storygraph_scraper.py** contiene el flujo de web scraping definitivo. Posee una estructura similar a la biblioteca: los métodos `book_search` (véase el código A.1) y `book_info` (véase el código A.2) tienen el mismo objetivo que los dos métodos mencionados anteriormente.

Además, `book_info` devuelve información más completa sobre los libros (véase la Figura 4.2). Destaca el atributo 'tag_percent', el cual no solo recopila más etiquetas de estado de ánimo que en su iteración anterior, sino que muestra el porcentaje de afinidad de cada etiqueta (un número de 0 a 100), información que ya se ha reconocido como importante en capítulos anteriores.

```
data = {
    # we need to 'get tags' from tag_percent (over 30?)
    'storygraph_id': id,
    'title': title,
    'authors': authors,
    'ISBN/UID': book_id,
    'pages': pages,
    'first_pub': first_pub,
    'tags': tags,
    'description': description,
    'cover-source': img,
    'n_reviews': n_reviews,
    'tag_percent': tag_percent,
    'pace_percent': pace_percent,
}
```

Figura 4.2: Objeto JSON que almacena la información definitiva de un libro de The StoryGraph.

En la BBDD final hay 635 libros de The StoryGraph.

4.1.2. Sinónimos: Thesaurus

Similarmente a la sección de libros, inicialmente se usó la biblioteca de Python `PyMultiDictionary` para extraer los sinónimos de Thesaurus de cada etiqueta. Se decidió poco después que sería más útil hacer un web scraping manual, con el propósito de conseguir la información requerida de forma personalizada. El código definitivo del scraping se encuentra en el fichero **scrapers/dictionary_scraper.py**, en el que destaca la función `thesaurus` (véase el código A.3).

El fichero accede a la BBDD y realiza una consulta para obtener las 14 etiquetas. Los sinónimos obtenidos de cada etiqueta se clasifican de acuerdo a las etiquetas que Thesaurus otorga a cada sinónimo en función de su afinidad. Los términos de la clasificación son ‘strongest’, ‘strong’ y ‘weak’.

etiqueta	número_de_sinónimos
adventurous	19
challenging	18
dark	47
emotional	48
funny	42
hopeful	34
informative	20
inspiring	16
lighthearted	34
mysterious	46
reflective	11
relaxing	44
sad	49
tense	7

(a) Número de sinónimos que le corresponde a cada etiqueta de The StoryGraph.

tipo_de_sinónimo	cantidad
SELF	14
THESAURUS	398

(b) Número de sinónimos en la BBDD, diferenciando entre los 14 términos inciales y los extraídos de Thesaurus.

Figura 4.3: Consultas a la BBDD sobre sinónimos de Thesaurus.

Una vez llena la BBDD con los sinónimos, la Figura 4.3(a) muestra cuántos sinónimos tiene cada etiqueta. El número total de sinónimos y, por tanto, el número de términos que se usarán en la búsqueda de música, es 412. El número de sinónimos excluyendo los 14 términos iniciales y, por tanto,

provenientes de Thesaurus, es 398 (véase la Figura 4.3(b)).

4.1.3. Música: LastFM

LastFM posee una API que permite obtener las top entidades de música (canciones, álbumes y artistas) asociadas a etiquetas de estado de ánimo que forman parte de su folksonomía. Para este trabajo se interactuará con dicha API no directamente, sino a través de la biblioteca `pylast`.

El flujo para obtener canciones comienza con la función `get_tag`, con la que se obtiene el objeto `Tag` (etiqueta) a partir de una cadena de texto. El objeto `Tag` tiene tres métodos principales para obtener canciones asociadas: el método `Tag.get_top_tracks(limit)` consigue las canciones más etiquetadas, poniendo como límite el número indicado por `limit`. Los métodos `Tag.get_top_albums(limit)` y `Tag.get_top_artists(limit)` operan similarmente, solo que obteniendo los álbumes y los artistas más etiquetados, respectivamente.

Se consideró importante aprovechar las tres vías para conseguir canciones. Por supuesto, la vía directa es conseguir las canciones más etiquetadas. Sin embargo, supone un flujo distinto que se recomiende una canción de un artista que está dentro de los artistas más etiquetados, y lo mismo ocurre con los álbumes. Es una decisión que añade variabilidad al algoritmo.

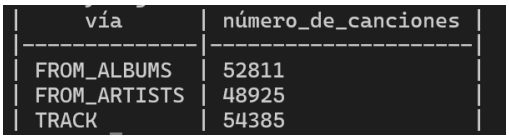
Así pues, una vez obtenida la lista de artistas, se utiliza el método `Artist.get_top_tracks(limit)` para obtener una lista de canciones asociadas a cada artista. Similarmente, una vez obtenida la lista de álbumes, se usa el método `Album.get_tracks`.

El fichero **scrapers/lastfm_scraper.py** contiene el código definitivo para extraer la música de LastFM. Este contiene una función llamada `get_lastfm_music(tag_list, network, n)` (véase el código A.4). El primer argumento es una lista de etiquetas sobre las que se hará una búsqueda en LastFM. El segundo argumento hace referencia a las credenciales de la API. El tercer argumento determinará el número de canciones que se intentará conseguir.

Se quiere conseguir aproximadamente el mismo número de canciones en cada una de las vías. Para conseguir esto, se utiliza el parámetro `n` de la función para estimar proporciones. Se obtendrán $10 * n$ canciones por vía directa. Por la vía de artistas, se obtendrán n artistas, y 10 canciones a partir de cada artista. Por la vía de álbumes, se obtendrán n álbumes, y todas las canciones de cada álbum.

Como siempre, el resultado se guarda en un fichero JSON. Al guardarse a continuación en la BBDD, como ya se mostró en el capítulo de diseño, se guarda cada asociación etiqueta-canción en función de la vía usada para conseguirla. Se le da el nombre `TRACK` si se consigue por vía directa, `FROM_ARTIST` si proviene de un artista y `FROM_ALBUM` si proviene de un álbum.

Se ha llenado la BBDD con un total de 153 448 canciones de LastFM, las cuales se dividen con relativa equidad entre las tres vías (véase la Figura 4.4).



vía	número_de_canciones
FROM_ALBUMS	52811
FROM_ARTISTS	48925
TRACK	54385

Figura 4.4: Consulta con el número de canciones de LastFM agrupadas por las tres vías para obtenerlas.

4.1.4. Música: Spotify

El último paso para conseguir un algoritmo funcional consiste en obtener, para cada canción, un enlace para poder escucharla. Se usará la Web API de Spotify para conseguir este objetivo. Concretamente, se hará uso de la función `search` de la API: una petición `GET` con dos parámetros a destacar. El parámetro de consulta es una cadena de texto para buscar la canción requerida (se incluyeron tanto el nombre de la canción como el nombre del artista). El parámetro límite determina el número de canciones devueltas; se estableció en 10 canciones.

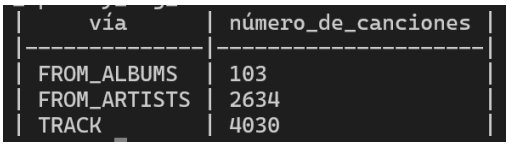
El fichero **scrapers/spotify_filter.py** contiene el código para obtener las canciones de Spotify. Destaca la función `spotify_search` (véase el código A.5). Se utiliza como parámetro de entrada el fichero JSON con las canciones de LastFM, y se rellena otro fichero JSON con las canciones de Spotify de salida.

Aquí se encontró un obstáculo importante: el método `search` solamente encontraba la canción buscada entre un 50 % y un 60 % de las veces, aunque se especificara la canción y el artista en el parámetro de consulta. Esto quiere decir que casi la mitad de las consultas devolvían una lista de 10 canciones, entre las cuales no se encontraba la canción buscada. Este problema ocurría incluso con canciones muy famosas que, evidentemente, existen en Spotify, por lo que se achaca a un problema del propio método.

Otro problema, y, probablemente, el más importante de todos, tiene que ver con el límite de solicitudes del método `search`: el sistema lograba completar 500 solicitudes aproximadamente antes de poner un timeout de 24 horas, durante las cuales la API quedaba inutilizable. Para superar este obstáculo, diariamente durante aproximadamente 20 días se obtuvieron las canciones en pequeños grupos.

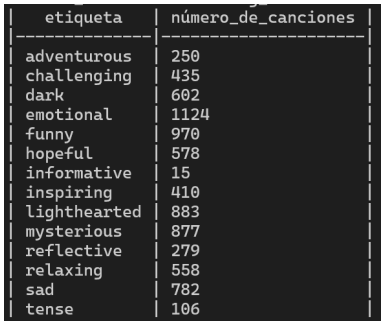
No es difícil ver que, combinando ambos problemas, el número final de canciones en la BBDD se ha reducido dramáticamente. Específicamente, a 6719 canciones. Con menos restricciones y más tiempo, este número podría haber aumentado más. Sin embargo, como cada usuario, al probar el algoritmo, obtendrá no más de 30 canciones, no hay razón para pensar que su experiencia se verá afectada.

La Figura 4.5(a) muestra las canciones de Spotify agrupadas por las tres vías de obtención. Se observa que la vía de álbumes es bastante más escueta que el resto, lo cual no se debe a ningún fallo estructural, sino a la falta de tiempo para recopilar más.



vía	número_de_canciones
FROM_ALBUMS	103
FROM_ARTISTS	2634
TRACK	4030

(a) Número de canciones de Spotify agrupadas por las tres vías para obtenerlas.



etiqueta	número_de_canciones
adventurous	250
challenging	435
dark	602
emotional	1124
funny	970
hopeful	578
informative	15
inspiring	410
lighthearted	883
mysterious	877
reflective	279
relaxing	558
sad	782
tense	106

(b) Número de canciones de Spotify que tiene cada etiqueta de The StoryGraph.

Figura 4.5: Consultas a la BBDD sobre canciones de Spotify.

Se termina esta sección con la Figura 4.5(b), la cual muestra las canciones de Spotify agrupadas en función de la etiqueta de The StoryGraph de la que provienen. Es notable la etiqueta 'informative', la cual tiene solamente 15 canciones; no es sorprendente debido a que es la única etiqueta de las catorce que no es exactamente un estado de ánimo, y no ha interactuado de la mejor forma con la música.

4.2. Página web

Como se estableció en el análisis de requisitos, se crea para la página web un proyecto de Django. Se creó el proyecto usando el comando `django-admin startproject tfg_webpage`. Uno de los ficheros que se crean automáticamente es **settings.py** (véase el código B.1), el cual contiene algunas configuraciones importantes del proyecto. Otro fichero auto-generado relevante al proyecto es el de **urls.py** (véase el código B.2), el cual incluirá todas las rutas relevantes para este proyecto, especialmente las de la API. Por último, destaca el fichero **manage.py**, el cual proporciona utilidad en la línea de comandos y será muy utilizado.

4.2.1. Back-end

El back-end, evidentemente, se realizará en Django. Una vez dentro del proyecto, se crea una nueva app llamada **webapp** con el comando `python3 manage.py startapp webapp`. Es importante registrar la app creada en el apartado `INSTALLED_APPS` de **settings.py**.

La BBDD, que en la Sección 4.1 se ha ido llenando con todas las entidades relevantes del algoritmo 'Book-to-Soundtrack', se ha creado gracias al fichero **models.py** (véase el código B.3). En este archivo se encuentra la ORM que facilita Django: cada modelo creado migra a una tabla de la BBDD (usando los comandos `makemigrations` y `migrate`).

En el proyecto se ha creado una carpeta que contiene los ficheros JSON con todos los datos de la BBDD. Luego, a través de un programa se rellena la BBDD a partir de dichos archivos JSON.

Hay dos ficheros Python relevantes para la API. El primero es **storygraph_scraper.py**, que contiene el mismo código que el fichero del mismo nombre de la carpeta **scrapers**, el cual recibe como entrada una cadena de texto que identifica un libro y devuelve un objeto JSON con su información de The StoryGraph. Es relevante en el modo en línea del algoritmo. El segundo fichero es **get_track_pool.py** (véase el código B.4), el cual, en función del libro de entrada y usando el algoritmo 'Book-to-Soundtrack', devuelve el conjunto de canciones asociadas a dicho libro.

4.2.2. API REST

La API se realiza también en Django, como extensión natural del back-end. Para montarla, se hace uso de la aplicación `rest_framework`. El primer uso que se le da a `rest_framework` es importar un enrutador en el fichero **urls.py**, el cual agrupa todas las rutas en la ruta `api/v1/` y les da una estructura unificada y organizada.

Por supuesto, una ruta no sería nada sin un controlador; es decir, la función que se activa cada vez que se llama a la ruta. Dichos controladores se encuentran en **views.py** (véase el código B.5). Todos los controladores tienen una estructura de clase y, además, extienden un objeto `viewset`, también importado de `rest_framework`.

Controlador	Ruta	Métodos API soportados
`BooksDB`	`books/`	[list, retrieve]
`GetBookByISBNorUID`	`isbn/`	[create]
`StorygraphSearch`	`storygraph-search/`	[create]
`getTagSynonyms`	`tag/`	[retrieve]
`getTrackPool`	`track-pool/`	[create]

Tabla 4.1: Tabla en la que se explican los distintos controladores en **views.py**.

La lista de todos los controladores de la API se muestra en la Tabla 4.1. `BooksDB` extiende un `viewset.ReadOnlyModelViewSet`, por lo que se implementan por defecto solo list y retrieve, los métodos de lectura. El queryset modelado es `Storygraph_book`, es decir, los libros almacenados en la BBDD. El resto de los controladores extienden `viewset.ViewSet`, el cual es más genérico.

`GetBookByISBNorUID` devuelve información del libro de The StoryGraph solicitado. Discrimina entre el modo en línea, en el cual se utiliza el fichero **scripts/storygraph_scraper**, y el modo fuera de línea, que lo busca en la BBDD. `StorygraphSearch` tiene una función similar al modo en línea del controlador anterior.

`getTagSynonyms` devuelve de la BBDD una lista de sinónimos asociados a la etiqueta enviada.

Por último, `getTrackPool` utiliza el fichero **scripts/get_track_pool.py** para obtener el conjunto de canciones de la BBDD asociadas al libro de entrada.

4.2.3. Front-end

En un principio se consideró implementar vistas en Django, lo cual simplificaría el proyecto y eliminaría la necesidad de una API. Sin embargo, el resultado sería demasiado estático y simple.

Rápidamente se decidió utilizar el framework React (con JavaScript) para crear la aplicación front-end, y desde cero con el build tool Vite. Se usó el comando `npm create vite@latest webapp-frontend-react` para crearla, eligiendo durante la configuración JS y React como opciones.

El único subdirectorio relevante es **webapp-frontend-react/src**, el cual contiene todo el código de la aplicación. Dentro, el fichero **App.jsx** (véase el código B.6) estructura la raíz del proyecto, utilizando un enrutador para organizar las rutas de la página web. En la tabla 4.2 se pueden ver todas las rutas asociadas a su componente de React. Se decidió que **el idioma utilizado en cada componente de la página web sería el inglés**.

Ruta	Componente de React
/	MainFlowView
/instructions	InstructionsView
/flow/choose-book	ChooseBookView
/flow/book/:mode/:book_id	SelectedBookView
/flow/book/ mode/ book_id/create-playlist	ConfigurePlaylistView
/flow/book/:mode/:book_id/show-playlist/:n_tracks	ShowPlaylistView
*	NotFoundView

Tabla 4.2: Tabla que muestra las distintas rutas de la página web asociadas a su componente de React.

A continuación, se ofrece una breve descripción de cada componente. `MainFlowView` es la ruta principal de la página web, la cual explica brevemente el propósito del proyecto y cómo funciona. Ofrece dos vías: se puede ir a `InstructionsView`, la ruta que contiene toda la información de las pruebas que se están realizando para este Trabajo de Fin de Grado, así como enlaces a los dos cuestionarios (véase el capítulo de pruebas para más información).

La segunda vía es, en efecto, la vía del algoritmo 'Book-to-Soundtrack' implementado. Se llega a `ChooseBookView`, donde el usuario elige el libro que quiere para su experiencia; efectivamente, se ofrece tanto el modo en línea como el modo fuera de línea. Elegir el libro deseado lleva a `SelectedBookView`, que muestra los detalles del libro elegido, la proporción de etiquetas de estado de ánimo del libro, y la lista de sinónimos relevante a cada etiqueta. En definitiva, te enseña el resumen emotivo que, en unos momentos, se transformará en una listade reproducción.

Siguiendo adelante, el componente `ConfigurePlaylistView` pide al usuario elegir el número de canciones de su lista de reproducción. Una vez elegida, el último componente, `ShowPlaylistView`, muestra al usuario el conjunto de todas las canciones recuperadas de la BBDD, no solo el número solicitado. Esto es para que el usuario, con solo un click, pueda elegir de ese conjunto inicial un subconjunto aleatorio del número de canciones deseado. Esta decisión de diseño pretende mejorar la experiencia del usuario.

Una limitación en la experiencia de usuario es que la lista de reproducción de música no se genera sola. Existe una forma, gracias a la API de Spotify, de autenticarse en tu propia cuenta y registrarla directamente. Sin embargo, no se llegó a implementar por limitaciones de tiempo. Alternativamente, como ya se ha mencionado, cada canción, al hacer click, abre su propio link de Spotify, y es tarea del usuario crear su propia lista de reproducción y añadir las canciones.

4.2.4. Despliegue en producción

Al tener dos aplicaciones distintas, el back-end y el front-end, también se necesitarán dos despliegues distintos de cada aplicación. Se utilizará Render para ello.

Para el back-end, se realizó el despliegue de un Web Service (servicio web). En la Figura 4.6 se puede ver la diferenciación entre el modo en desarrollo y el modo en producción. El modo en producción obtiene las variables `SECRET_KEY` y `ALLOWED_HOSTS` de unas variables de entorno que se configuran en el entorno de Render.

```
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = os.getenv('DEBUG') # should be 1
if DEBUG == None:
    DEBUG = True
    SECRET_KEY = 'django-insecure-fik(y#q7)kseaiqh+k8$g8w&itx@8p3v8)2abq8a8xn$r)y**='
    ALLOWED_HOSTS = ['*']
else:
    SECRET_KEY = os.getenv('SECRET_KEY')
    ALLOWED_HOSTS = os.getenv('ALLOWED_HOSTS').split(' ')
```

Figura 4.6: Diferenciación entre el modo en desarrollo y en producción usado en **settings.py**.

Otras formas de configuración incluyen: el fichero **requirements.txt** (véase el fichero B.7) con las dependencias del proyecto back-end y el fichero **build_production.sh** (véase el fichero B.8), el cual se ejecuta al principio y que, entre otras cosas, instala las dependencias de **requirements.txt** en el servicio web de Render. El comando de despliegue del servicio web es `gunicorn tfg_webpage.wsgi:application`.

Para el front-end, se despliega una static site (página estática). El comando que empaqueta la aplicación para ser utilizada es `npm run build`. Las dependencias del proyecto están incluidas en el fichero **package.json** (véase el fichero B.9). La URL publicada por Render y que se usó para acceder a la página web es `https://tfg-book-to-soundtrack-site.onrender.com`.

5

Pruebas

Las pruebas constituyen una parte importante del trabajo, pues validarán el producto creado. Se dividen en dos cuestionarios que los usuarios deben completar en orden.

El primer cuestionario es pertinente a la usabilidad de la página web, y acompaña al usuario mientras que éste utiliza sus varias herramientas. En este proceso, el usuario probará de principio a fin el algoritmo 'Book-to-Soundtrack', por lo que acabará con una lista de reproducción de música asociada a un libro de su elección. Por último, contestará a unas preguntas finales sobre la página web.

El segundo cuestionario es pertinente a la lista de reproducción de música que los usuarios han recogido. Su tarea, para la cual recibieron una semana aproximadamente, es escuchar su lista de reproducción. Una vez hecho eso, rellenan el cuestionario, el cual tiene preguntas relacionadas a la música que han escuchado.

Antes que nada, los usuarios en ambos cuestionarios tuvieron que responder una pregunta de consentimiento, en la que confirman que entienden la información explicada al principio del cuestionario, y que participan 'de forma libre, anónima y voluntaria en este estudio'. Como es evidente, todo usuario que registró su respuesta dio su consentimiento en ambos cuestionarios.

Luego, los usuarios respondieron varias preguntas sobre demografía. Aunque tuvieron que rellenar las mismas preguntas en cuestionarios distintos las proporciones de respuestas en temas de género y edad son prácticamente idénticos. La Figura 5.1(a) muestra la edad de los participantes, y se observa que la gran mayoría de participantes tienen entre 18 y 29 años, con una pequeña minoría entre 48 y 65 años. La Figura 5.1(b) muestra el género de los participantes, la cual muestra que todos los participantes se encuentran en el binario de género. Además, aproximadamente dos tercios de los participantes son mujeres.

5.1. Primer cuestionario: usabilidad de la página web

Se recibieron un total de 16 respuestas en esta encuesta. La primera tarea que los participantes realizaron en la página web fue buscar un libro entre los 635 guardados en la BBDD, es decir, interac-

(a) Pregunta sobre la edad de los participantes.

(b) Pregunta sobre el género de los participantes.

Figura 5.1: Preguntas sobre edad y género de los participantes.

tuaron con el modo fuera de línea. Concretamente, debido a que la lista está ordenada por nombre de autor, se les pidió buscar un libro cuyo autor comenzara por **una letra entre la G y la T** (véase la Figura 5.2 para observar la distribución de las letras entre los participantes). Este intervalo del abecedario es deliberado, pues fuerza a los participantes a recorrer la lista.

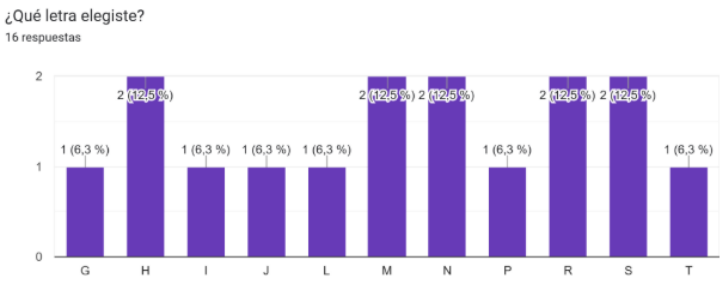


Figura 5.2: Distribución de la letra elegida para la búsqueda de un libro fuera de línea.

Esta primera búsqueda se consideró generalmente fácil de hacer. La primera pregunta, '¿Cómo de fácil fue encontrar el libro?' (véase la Figura 5.3(a)), tuvo una respuesta mayoritariamente positiva, con un total del 68,8 % de los participantes respondiendo 'Fácil' o 'Muy fácil', y de los cuales un 25 % respondió 'Muy fácil'. Solamente un 12,5 % de respuestas fueron negativas, y todas fueron 'Difícil', con cero participantes respondiendo 'Bastante difícil'.

La segunda pregunta, '¿Cómo de intuitiva/fácil de navegar fue la lista de libros?' (véase la Figura 5.3(b)), tuvo respuestas un tanto más positivas que la primera. Concretamente, un 37,5 % de los participantes respondieron 'Muy fácil/muy intuitiva', la respuesta más positiva. Un 81,3 % de participantes respondieron positivamente. El mismo porcentaje respondió negativamente, mientras que bajó la respuesta 'Neutral'.

Esto se debe a la forma en la que están formuladas las preguntas. La primera pregunta es más específica y pertinente a la propia búsqueda y, como resultado, una mayor proporción de participantes se consideró neutral. La segunda pregunta es más general y está dirigida a la propia herramienta de la página web, la cual ha causado una impresión general más positiva.

(a) Pregunta '¿Cómo de fácil fue encontrar el libro?'.

(b) Pregunta '¿Cómo de intuitiva/fácil de navegar fue la lista de libros?'.

Figura 5.3: Preguntas sobre el modo fuera de línea de la búsqueda de un libro.

La siguiente tarea corresponde a la búsqueda en línea de un libro, dejado a la elección de los participantes. El proceso involucra introducir información del libro en un buscador (se animó a los participantes a insertar el ISBN para mayor facilidad de búsqueda) y confirmar si la búsqueda era la correcta. En caso de que no lo sea, repetir el proceso hasta que lo encuentren.

La Figura 5.4(a) muestra la pregunta '¿Has podido encontrar tu libro deseado?', en la que todos los participantes respondieron 'Sí'. Más aún, en la pregunta '¿Cómo de fácil fue el proceso?' (véase la Figura 5.4(b)), un 75 % de participantes respondieron 'Muy fácil', y un 93,8 % de los paricipantes dieron respuestas positivas, siendo el resto neutrales.

(a) Pregunta '¿Has podido encontrar tu libro deseado?'.

(b) Pregunta '¿Cómo de fácil fue el proceso?'.

Figura 5.4: Preguntas sobre el modo en línea de la búsqueda de un libro.

La tercera tarea requiere que los participantes, a partir del libro de la tarea anterior, prosigan con la página web, pasando por las páginas de detalles del libro y la página donde eligen el número de canciones, para llegar a la página de la lista de reproducción. La tarea termina en cuanto guardan la lista de reproducción.

De nuevo, la experiencia general ha resultado positiva. La pregunta '¿El recorrido trazado por la página web desde tu libro hasta la música te ha resultado intuitivo?' (véase la Figura 5.5(a)) ha obtenido un 93,8 % de respuestas positivas. De éstas, los participantes han respondido 'Muy intuitivo' un 56,3 %. Un 6,2 % (una persona) ha contestado negativamente. Similarmente, a la pregunta '¿La información presentada a lo largo del recorrido se ha entendido?' (véase la Figura 5.5(b)) un 93,8 % de participantes respondieron positivamente, y un 68.8 % respondieron 'Sí, totalmente'.

(a) Pregunta '¿El recorrido trazado por la página web desde tu libro hasta la música te ha resultado intuitivo?'.

(b) Pregunta '¿La información presentada a lo largo del recorrido se ha entendido?'.

Figura 5.5: Preguntas sobre la navegación por la página web.

La última pregunta para finalizar esta tarea requiere a los participantes puntuar del 1 al 5 cuánto problema les ha supuesto los tiempos de espera y/o refrescar la página, siendo un 1 'Ningún problema' y un 5 'Mucho problema' (véase la Figura 5.6). Se consideró importante debido a los elevados tiempos de espera que experimentamos con la página web en producción, probablemente debido a la reducida capacidad de procesamiento que ofrece Render en su suscripción gratuita.

Sin embargo, la respuesta fue ampliamente positiva: un 93,8 % de participantes respondió positivamente, con un 50 % de ellos dando un 1. Un 6,3 % de personas respondió con un 3.

Figura 5.6: Pregunta 'Puntúa, del 1 al 5, cuánto problema han supuesto los tiempos de espera/refrescar la página'.

Una vez terminadas las tareas relacionadas con la página web, los participantes deben responder unas últimas preguntas de naturaleza general sobre la página web, así como rellenar una escala de usabilidad (SUS).

La pregunta '¿Te ha gustado el diseño de la página web?' (véase la Figura 5.7(a)) ha obtenido un 50 % de respuestas 'Sí, mucho', un 25 % de 'Sí' y otro 25 % de 'Neutral'. es, por tanto, una reacción ampliamente positiva.

La pregunta '¿Has disfrutado el concepto de la página web?' (véase la Figura 5.7(b)) recibe la respuesta más positiva hasta el momento: todos los participantes respondieron positivamente, con un 81,3 % respondiendo 'Sí, mucho'. Esto es porque la pregunta se refiere al concepto de la página web y del proyecto en general más que la ejecución, algo que, como puede verse, ha recibido una reacción extremadamente positiva.

(a) Pregunta '¿Te ha gustado el diseño de la página web?'.

(b) Pregunta '¿Has disfrutado el concepto de la página web?'.

Figura 5.7: Preguntas sobre la experiencia general con la página web.

La última pregunta antes de la escala de usabilidad es 'Puntúa la página web sobre 10', la cual se presenta en la tabla 5.1. Puede verse una recepción decididamente positiva: nadie le ha dado una puntuación del 1 al 6. Puede verse que la puntuación más popular ha sido un 9 (la moda). La mediana también es un 9. La media aritmética es 8,6875, un 8,69 si se aproxima a dos decimales.

Puntuación (x_i)	**F. absoluta (f_i)**	**F. relativa (h_i)**	**% (p_i)**	$x_i * h_i$
1-6	0	0	0	0
7	2	0,125	12,5	0,875
8	4	0,25	25	2
9	7	0,4375	43,75	3,9375
10	3	0,1875	18,75	1,875
Total	**16**	**1**	**100**	**8,6875** ($\bar{x}$)
Mediana ($\tilde{x}$)	**9**			
Moda ($\hat{x}$)	**9**			

Tabla 5.1: Tabla con la distribución de respuestas para la pregunta 'Puntúa la página web sobre 10'.

Se termina el análisis del primer cuestionario con una escala de usabilidad (SUS). Tras hacer los cálculos, la puntuación SUS del sistema, calificado de 0 a 100, es de 88,75, lo cual es una puntuación excelente. Se puede decir que los objetivos de usabilidad de la página web se han cumplido.

Antes de continuar con el siguiente cuestionario, hay algunos comentarios anónimos escritos por participantes que resultan interesantes y añaden críticas y comentarios que enriquecen el proyecto.

Muchos participantes, en sus comentarios, tienen sugerencias de mejora de la lista de libros en el modo fuera de línea. Una de ellas es que las pestañas númericas hacen un poco más difícil la búsqueda de un libro ordenado por letra: un participante propone que 'pestañas por [letras] A-B-C-...-Z de nombre de autor sería más user-friendly'. Otra sugerencia, esta vez de la búsqueda en línea de un libro, es que no devuelve una lista de libros, sino un único libro, y eso impacta la facilidad de uso. Otro usuario '[ha echado] de menos en Local Library más autores en castellano' (el modo fuera de línea).

La sugerencia más relevante concierne los enlaces de Spotify para escuchar la música. Un usuario

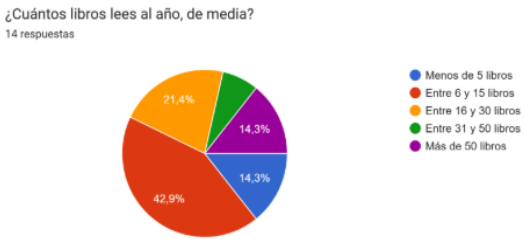
comenta que ‘molaría que la playlist se generara directamente en spotify y que tú puedas elegir simplemente guardarla’. Otro usuario propone una solución que nos ha parecido muy buena: a falta de exportación automática a Spotify por ser ‘muy tricky’ (muy difícil), ‘quizá sea asequible poder convertir la playlist en PDF, con sus enlaces y todo (nombre de canciones, libro al que está asociado y moods), para poder ser accesible en otro momento’. De esta forma, la persistencia de la lista de reproducción ya no es un problema y, más aún, el ‘viaje’ entero de la página web podría guardarse como un archivo PDF, para el beneficio del usuario.

5.2. Segundo cuestionario: lista de reproducción de música

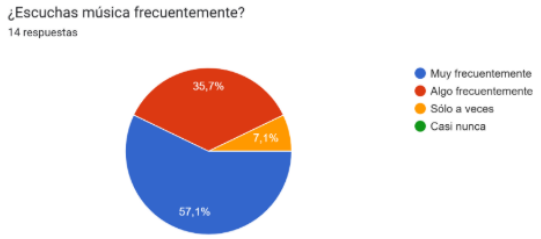
Se recibieron un total de 14 respuestas a esta encuesta, dos menos que la primera, lo cual implica que dos personas no completaron la segunda encuesta a tiempo.

El segundo cuestionario contiene preguntas específicas sobre la lista de reproducción de música que cada participante ha escuchado en preparación. Es aquí donde se analizará con detalle si el algoritmo ‘Book-to-Soundtrack’ ha tenido éxito, desde varias perspectivas.

Antes de nada, los participantes, en la sección de demografía, contestaron a unas preguntas extra sobre sus hábitos de lectura y de escucha de música. A la pregunta ‘¿Cuántos libros lees al año, de media?’ (véase la Figura 5.8(a)), la respuesta de los participantes fue bastante variada: un 14,3 % lee menos de 5 libros, un 42,9 % (la mayoría) lee entre 6 y 15, un 21,4 % lee entre 16 y 30, un 7,1 % lee entre 31 y 50 y un 14,2 % lee más de 50.



(a) Pregunta ‘¿Cuántos libros lees al año, de media?’.



(b) Pregunta ‘¿Escuchas música frecuentemente?’.

Figura 5.8: Preguntas sobre hábitos de lectura y escucha de música.

Esto es significativo, pues, cumulativamente, la gran mayoría (un 85,7 %) lee más de 5 libros y casi la mitad (un 42,8 %) lee más de quince. Es seguro afirmar que los participantes, en su conjunto, tienen hábito lector.

Ahora, comentemos los hábitos de escuchar música de los participantes. A la pregunta ‘¿Escuchas música frecuentemente?’ (véase la Figura 5.8(b)) el 57,1 % respondió ‘Muy frecuentemente’, el 35 % respondió ‘Algo frecuentemente’, el 7,1 % respondió ‘Sólo a veces’ y nadie respondió ‘Casi nunca’.

Parece ser que el conjunto de participantes sin duda pasa más tiempo escuchando música que leyendo. Esto no es sorprendente debido a que la lectura requiere un bastante más esfuerzo cognitivo que escuchar música, y para leer es necesario sostener la atención, cosa que es no es necesaria con la música.

Otra interesante pregunta es 'En los momentos que estás leyendo, ¿sueles escuchar música de acompañamiento?' (véase la Figura 5.9), en la que la respuesta mayoritaria de los participantes (con un 64,3 %) es 'Casi nunca'. Un 14,3 % respondió o 'Con poca frecuencia' y el resto minoritario se distribuyó equitativamente en el resto de respuestas.

Figura 5.9: Respuestas a la pregunta 'En los momentos que estás leyendo, ¿sueles escuchar música de acompañamiento?'.

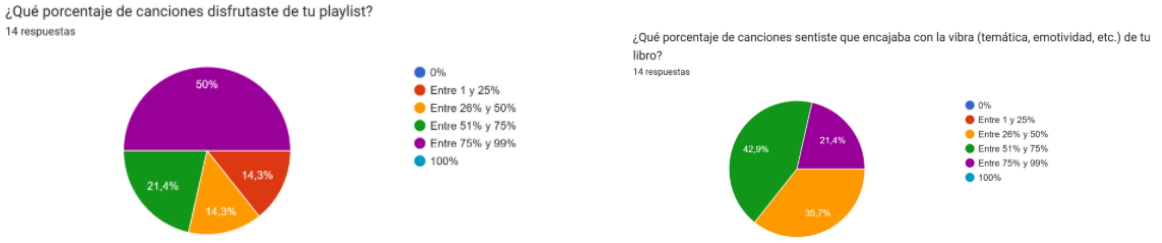
Esto refleja que, en este conjunto de de participantes, no existe un deseo particular de escuchar música mientras que se lee. Esto podría ser porque la música puede resultar una distracción o, simplemente, para la mayoría de participantes, la lectura y la música no son facetas que estén unidas. Esto podría revelar que el propósito normal para el que una persona usaría la herramienta 'Book-to-Soundtrack' no es para escuchar música mientras que se lee, lo cual no era el propósito principal del algoritmo.

Con esto se termina la sección introductoria y comienzan las preguntas de crítica del algoritmo. Comencemos por la más general, la pregunta '¿Te ha parecido una propuesta interesante/curiosa?', la cual ha tenido una respuesta extremadamente positiva: el 92,9 % de los participantes respondió 'Muy interesante/muy curiosa', y todas las respuestas son positivas (véase la Figura 5.10). Esto denota el reconocimiento de la innovación y la creatividad de este proyecto,

Figura 5.10: Respuestas a la pregunta '¿Te ha parecido una propuesta interesante/curiosa?'.

A la pregunta '¿Qué porcentaje de canciones disfrutaste de tu playlist?' ningún participante respon-

dió ‘0 %’ o ‘100 %’, algo esperado. Un 50 % de participantes respondió disfrutó más de tres cuartos de su lista, y un 71,4 % disfrutó la mitad o más de sus canciones. Solo un 14,3 % disfrutó menos de un cuarto (véase la Figura 5.11(a)).



(a) Pregunta ‘¿Qué porcentaje de canciones disfrutaste de tu playlist?’.

(b) Pregunta ‘¿Qué porcentaje de canciones sentías que encajaba con la vibra de tu libro?’.

Figura 5.11: Preguntas sobre la lista de reproducción.

Esto refleja un dato importante pues, a simple vista, parece una pregunta llena de subjetividad, pero si tomamos en cuenta que el libro que eligieron los participantes es uno que les entusiasma, disfrutan y se lo han leído, es un buen reflejo del algoritmo que, en su mayoría, haya impresiones mayoritariamente positivas sobre las canciones que han escuchado. Es algo invariablemente bueno que la gran mayoría de los participantes haya encontrado música nueva que les gusta a partir de esta herramienta que probablemente nunca habría encontrado de otra manera.

La siguiente pregunta es ‘¿Qué porcentaje de canciones sentías que encajaba con la vibra de tu libro?’. De nuevo, nadie respondió ‘0 %’ o ‘100 %’. Un 21,4 % de participantes sintió que tres cuartos o más de su lista encaja con su libro, y un 64,3 % lo sintió con la mitad, o más, de su lista (véase la Figura 5.11(b)).

Son respuestas en su mayoría positivas, lo cual es satisfactorio, pues se podría tratar esta pregunta como el objetivo principal de este trabajo.

La siguiente pregunta es ‘¿Usarías esta playlist para escuchar mientras lees tu libro?’, y un 64,3 % de participantes ha contestado que no (véase la Figura 5.12(a)). Aunque esto sea una respuesta negativa, no lo es completamente y, además, era esperada. Como ya se analizó en la Figura 5.9, no hay un interés generalizado en escuchar música mientras se lee, y el algoritmo no se ha refinado para ese fin.

Continuando, a la pregunta ‘¿Usarías esta playlist antes o después de leer tu libro para ponerte en situación o mantener el ambiente o la vibra del mismo?’ un 50 % ha contestado que sí (véase la Figura 5.12(b)).

Es de esperar, pues esta pregunta, al igual que la anterior, separa temporalmente las acciones de leer y de escuchar música. Es, aún así, cierto triunfo para el algoritmo que la mitad de los participantes respondieran positivamente.

Cabe añadir un comentario anónimo de un participante, que dice: ‘Creo que usaría la playlist, más

(a) Pregunta '¿Usarías esta playlist para escuchar mientras lees tu libro?'.

(b) Pregunta '¿Usarías esta playlist antes o después de leer tu libro..?'.

Figura 5.12: Preguntas sobre la relación entre lista de reproducción y el libro.

que para ponerme en el mood, después de haber terminado el libro para recordarlo', una faceta de la herramienta que no está reflejada en este cuestionario, pero que es un uso perfectamente válido de ella. En efecto, esto revela una característica del algoritmo: que los usos que se le pueden dar son variopintos y dependientes de la subjetividad del usuario.

Resaltan otros comentarios anónimos interesantes en sus críticas y comentarios sobre el proyecto. Dos participantes lamentan la falta de música en español, ya que les habría parecido más acorde a su libro elegido. En efecto, al extraer de LastFM indiscriminadamente, la gran mayoría de canciones son en el idioma inglés. Otro participante cree que sería buena idea 'si se desarrolla más el proyecto [y se conecta] a más plataformas de streaming', pues pese a no tener Spotify, pudo encontrar todas las canciones propuestas en su plataforma preferida.

La tabla 5.2 muestra la distribución de respuestas para la pregunta 'Puntúa, del 1 al 5, cuán acertada es la playlist de música generalmente en relación con tu libro'. Como se puede observar, tanto la mediana como la moda son un 4, y la media es un 3,643 sobre 5, o un 7,286 sobre 10.

Puntuación (x_i)	**F. absoluta(f_i)**	**F. relativa (h_i)**	**% (p_i)**	$x_i * h_i$
1	0	0	0	0
2	1	0,0714	7,14	0,1428
3	5	0,3571	35,71	1,0713
4	6	0,4286	42,86	1,7144
5	2	0,1429	14,29	0,7145
Total	**14**	**1**	**100**	**3,643 ($\bar{x}$)**
Mediana ($\tilde{x}$)	**4**			
Moda ($\hat{x}$)	**4**			

Tabla 5.2: Tabla con la distribución de respuestas para la pregunta 'Puntúa, del 1 al 5, cuán acertada es la playlist de música generalmente en relación con tu libro'.

Antes de terminar esta sección y capítulo es interesante detenerse en dos comentarios anónimos muy detallados, uno positivo y uno negativo, que dos usuarios ofrecieron sobre su experiencia general

con la herramienta ‘Book-to-Soundtrack’. Creemos que contrastarlos ofrecerá un buen cierre.

Comencemos por el comentario negativo de un participante que eligió ‘Percy Jackson’ como su libro. Aunque hubo algunas canciones que ‘sí que pegaban’ e incluso se guardó una para su lista personal, encontró que ‘Había muchas canciones que no pegaban en general, de géneros muy dispares, e incluso me recomendó dos audios que eran grabaciones de dos shows de stand-up comedy’, así como ‘varias de Death Metal y una de Rock cristiano’. Considera que la idea del proyecto es muy buena, pero que se necesitan más adjetivos para describir mejor el tipo de canciones que se recomiendan, citando ‘mitológico’ como un ejemplo para Percy Jackson que habría funcionado mejor. A pesar de todo esto, insiste en que ‘no lo clasificaría como un fracaso para nada’.

Esto revela una limitación muy grande de la herramienta, pues todas las etiquetas utilizadas para unir libros con música son estados de ánimo. Probablemente, incluyendo etiquetas que reflejaran aspectos distintos de los libros, como descriptores de la estética del libro, habrían llevado a un experimento más exitoso para este participante. Por otro lado, comentar que los shows de stand-up que menciona el participante probablemente surgieron a partir de la etiqueta ‘funny’ o uno de sus sinónimos; es cierto que los resultados de esa categoría de música son polarizantes.

Por otro lado, el participante del comentario positivo, quien eligió ‘Hamnet’ como su libro, describe que, de su lista de 10 canciones, ‘había unas 7 canciones que encajaban perfectamente con la emocionalidad, delicadeza y sutileza del libro (por letra o por musicalidad)’, y que considera que no solo la herramienta le permite conocer nueva música, sino que le permite ‘re-sumergirse en ciertas escenas del libro o en sus imágenes a través de la música’. Concluye diciendo que usará la herramienta en sus próximas lecturas.

Este comentario resume todos los objetivos que se querían alcanzar con este trabajo: conseguir una unión entre libro y música, evocar escenas del libro a partir de la escucha de una canción y demostrar que la herramienta tiene valor para usarse de forma continuada y no solamente una vez.

Se termina con una última comparación entre los dos libros mencionados: mientras que ‘Hamnet’ es un libro ‘literario’, en el cual hay mucho énfasis en emocionalidad profunda, ‘Percy Jackson’ es una novela juvenil de acción con estética mitológica. La diferencia entre estos dos libros es evidente: el primero se puede describir mucho mejor, o incluso únicamente, por su emocionalidad, mientras que para el segundo resulta una descripción insuficiente que necesitaría de categorías de etiquetas como ‘estética’ o ‘aspecto’. Es interesante, de esta forma, confirmar que la taxonomía de libros que ofrece The StoryGraph es más apta para describir libros literarios o que priorizan una emocionalidad rica y diversa.

6

Conclusiones y trabajo futuro

Tras haber analizado las pruebas del algoritmo, así como los comentarios anónimos de los participantes, se puede concluir que el proyecto ha cumplido con éxito los objetivos que se planteaba. Generalmente, la experiencia de los usuarios con sus listas de reproducción de música se encontraba entre positiva y muy positiva, con un porcentaje más pequeño de experiencia neutral.

Con diferencia, el resultado más negativo de las pruebas es el de si escucharías la música mientras que lees. Esto se interpretó como que los participantes, en general, no estaban interesados en escuchar música mientras que leen. Sin embargo, también refleja que el algoritmo 'Book-to-Soundtrack', como se ha desarrollado en este proyecto, no tenía ese objetivo en mente. Una de las ideas que se tuvo al principio, y que, aunque no llegó a implementarse por las limitaciones del Trabajo de Fin de Grado, puede reservarse como trabajo futuro, es usar la API de Spotify para seleccionar canciones con un cierto tempo o velocidad. De esta forma se podría elegir música más lenta y calmada para el propósito de ser escuchada con la lectura. Otro atributo considerado importante para este fin sería seleccionar de manera explícita música sin letra.

El propósito de este trabajo, más bien, era proporcionar a un usuario música que 'le recordara' a su libro, que 'encajara con' su libro, que le recordara a su libro cuando la escuchara; una tarea que parece contradictoria a proporcionar música que permita escucharse mientras se lee. Combinar ambas tareas sería, desde luego, una buena ampliación de este trabajo.

También ha sido interesante intentar sintetizar el estado de ánimo o la emocionalidad de un libro a través de etiquetas descriptivas, para que luego eso se traduzca en buenas recomendaciones de música. Consideramos que ha funcionado muy positivamente en muchos casos, como se mencionó con el libro 'Hamnet' y otros libros similares a este. Sin embargo, en otros casos ha sido una solución insuficiente, y probablemente el algoritmo se beneficiaría de incorporar más categorías de etiquetas. Se deja como futuro trabajo incorporarlas a la herramienta, buscándolas en plataformas más allá de The StoryGraph.

Fueron varios participantes los que mencionaron que la herramienta se beneficiaría de un conjunto de datos más amplio de libros y canciones. Los libros no han supuesto un problema, puesto que ha sido posible incorporar el modo en línea, lo cual abre la búsqueda a prácticamente cualquier libro,

siempre y cuando se encuentre en la plataforma The StoryGraph. Sin embargo, la parte de música es la que no ha cumplido con los objetivos pensados: se ha pasado de 150,000 canciones recopiladas de LastFM a 6,500 debido al funcionamiento limitado de la API de Spotify. Creo, sin embargo, que una vez más revela el potencial del algoritmo, ya que, incluso con una versión degenerada del dataset de música, ha habido una mayoría de casos exitosos. Otro punto ciego es el de incorporar más música en el idioma español; tal vez el algoritmo se beneficiaría de permitir al usuario elegir el idioma de la música que desea.

Por otro lado, la página web como proyecto full-stack también fue un éxito. La crítica que, de nuevo, varios participantes mencionaron y que destaca, es la de permitir algún tipo de persistencia de la lista de reproducción creada. Se habló sobre implementar un sistema de registro y login, el cual permitiría a un usuario guardar lo que ha hecho en su perfil. La sugerencia anónima de, en cambio, guardarlo en un PDF también la consideramos muy buena y, además, no requeriría registro en la web.

Una interesante ampliación de este proyecto es basar un sistema de recomendación en él. De nuevo, aunque este proyecto es adyacente a los sistemas de recomendación, tal vez sea más acertado denominarlo un ‘sistema personalizado de sugerencias’, por ejemplo. Se podrían recopilar datos de usuario, los cuales se podrían usar para construir un sistema más sofisticado de predicción y recomendaciones, en base a las interacciones continuas de los usuarios con el sistema, para ir aprendiendo y mejorando las mismas. Esto también permitiría mejorar el mapeo entre etiquetas emocionales de Storygraph y LastFM, que componen el núcleo del algoritmo.

Como conclusión, este proyecto nació de la idea expuesta en la introducción: existe una unión entre la literatura y la música que se puede cuantificar y automatizar con herramientas informáticas. Creemos que, ante esta tarea, este trabajo es una aportación valiosa y expande las posibilidades de cómo se puede pensar en ambas artes. Y lo más importante, te lleva a reflexionar sobre tu relación con ellos.

Bibliografía

[1] D. G. Huamán Acuña, "Sistema de recomendación de libros basado en ontologías asociadas a tesauros: el caso de la biblioteca de la facultad de ingeniería de sistemas e informática de la universidad nacional mayor de san marcos," 2019.

[2] M. B. Lopez, A. J. H. Montes, R. V. Ramírez, G. A. Hernández, R. Z. Cabada, and M. L. B. Estrada, "Emoremsys: Sistema de recomendación de recursos educativos basado en detección de emociones/emoremsys: An educational recommender system by using emotions detection," *Revista Ibérica de Sistemas e Tecnologias de Informação*, no. 17, p. 80, 2016.

[3] J. Wang, A. Bellogín, and I. Cantador, "Integrating sentiment features in factorization machines: Experiments on music recommender systems," in *Proceedings of the 32nd ACM Conference on User Modeling, Adaptation and Personalization, UMAP 2024, Cagliari, Italy, July 1-4, 2024*, pp. 183–188, ACM, 2024.

[4] L. Betarte, R. Machado, and V. Molina, "Pgmusica: sistema de recomendación de música basado en filtrado colaborativo," 2006.

[5] A. M. Calvo, V. L. Batista, and G. V. González, "Sistema de recomendación de música basado en etiquetado social," *AVANCES EN INFORMÁTICA Y AUTOMÁTICA*, vol. 47.

[6] M. F. Dacrema, I. Cantador, I. Fernández-Tobías, S. Berkovsky, and P. Cremonesi, "Design and evaluation of cross-domain recommender systems," in *Recommender Systems Handbook* (F. Ricci, L. Rokach, and B. Shapira, eds.), pp. 485–516, Springer US, 2022.

[7] M. Sordo, F. Gouyon, L. Sarmento, Ò. Celma, and X. Serra, "Inferring semantic facets of a music folksonomy with wikipedia," *Journal of New Music Research*, vol. 42, no. 4, pp. 346–363, 2013.

[8] I. Fernández-Tobías, I. Cantador, and L. Plaza, "A social tag-based dimensional model of emotions: Building cross-domain folksonomies," *Procesamiento del lenguaje natural*, no. 51, pp. 195–202, 2013.

[9] E. Çano, M. Morisio, *et al.*, "Music mood dataset creation based on last. fm tags," in *2017 International Conference on Artificial Intelligence and Applications, Vienna, Austria*, pp. 15–26, 2017.

[10] A. Guyot, *Understanding Booksonomies: How and why are book taggers tagging?* PhD thesis, Aberystwyth University, 2013.

Apéndices

A

Scrapers

Este capítulo compila el código de los ficheros que conforman el algoritmo 'Book-to-Soundtrack', ya sea por ser usado en el back-end para conseguir información para el modo en línea, como por ser utilizados para recopilar datos para la BBDD.

Código A.1: Código correspondiente de la función `book_search` de **storygraph_scraper.py**.

```
1  def book_search(search_list, check_if_found = False):
2
3      parsed_list = []
4      for term in search_list:
5          parsed_term = re.sub("\+", "␣", term)
6          parsed_list.append(parsed_term)
7      print(parsed_list)
8
9      url_str = ''
10     for term in parsed_list:
11         # console_str += f'| {term} '
12         parsed_term = re.sub("␣", "%20", term)
13         url_str += f'{parsed_term}%20'
14
15     url = f'https://app.thestorygraph.com/browse?search_term={url_str}'
16
17     # request for the webpage of a certain book
18     req = Request(
19         url=url,
20         headers={"authority": "www.google.com",
21         "accept": "text/html,application/xhtml+xml,application/xml",
22         "accept-language": "en-US,en;q=0.9",
23         "referer": "www.google.com",
24         "user-agent": "Mozilla/5.0␣(Windows␣NT␣10.0;␣Win64;␣x64)␣AppleWebKit/537.36␣(KHTML,␣
               like␣Gecko)␣Chrome/129.0.0.0␣Safari/537.36"}
25     )
26
27     try:
28         response = urlopen(req)
29         time.sleep(0.1)
```

```
            print(response.status, response.reason)

            webpage = response.read()
            soup = bs(webpage, 'html.parser')

            id = soup.find('div', class_='max-w-3xl mb-4 md:mb-6 text-darkestGrey dark:text-grey
                book-pane break-words').get_attribute_list('data-book-id')[0]
            if check_if_found:
                title = soup.find('h3', class_='font-bold text-xl').find('a').text
                author = soup.find('p', class_='font-body font-normal text-xs').find('a').text
                found = parsed_list[0] in title and parsed_list[1] in author
                if found:
                    print('Match found!')
                else:
                    print('Not found..')

    except Exception as e:
            print(e)
            if check_if_found:
                print('Not found..')
                return (None, False)
            else:
                return None

    if check_if_found:
            return (id, found)
    else:
            return id
```

Código A.2: Código correspondiente a la función `book_info` de **storygraph_scraper.py**.

```
def book_info(id):

    url = f'https://app.thestorygraph.com/books/{id}'

    # request for the webpage of a certain book
    req = Request(
            url=url,
            headers={"authority": "www.google.com",
            "accept": "text/html,application/xhtml+xml,application/xml",
            "accept-language": "en-US,en;q=0.9",
            "referer": "www.google.com",
            "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
                like Gecko) Chrome/129.0.0.0 Safari/537.36"}
    )

    try:
            response = urlopen(req)
            time.sleep(0.1)
            print(response.status, response.reason)
```

```
19            webpage = response.read()
20            soup = bs(webpage, 'html.parser')
21
22            # parsing the title with regex
23            title = soup.find('h3',class_="font-semibold␣text-2xl␣md:w-11/12␣inline␣items-center").text
24            title = re.sub('␣␣+', '', title)
25            title = re.sub('\n', '', title)
26
27            # parsing authors
28            authors = []
29            auths = soup.find('p', class_='font-body␣font-medium␣mb-1␣text-base␣md:text-lg␣
                  md:w-11/12').find_all('a', href=True)
30            for a in auths:
31                authors.append(a.text)
32
33            # parse unique book identifier
34            book_id = soup.find('div', class_='hidden␣edition-info␣mt-3').find_all('p')[0].text.split('␣')[2]
35            # print(book_id)
36
37            # parsing pages and first year of publication
38            pages_first_pub = soup.find('p', class_='text-sm␣font-light␣text-darkestGrey␣dark:text-grey␣
                  mt-1').find_all('span')
39
40            pages = pages_first_pub[0].text.split('␣')[0]
41
42            first_pub = pages_first_pub[1].text.split('␣')[6]
43
44            # parsing tags
45            tags = []
46            tag_div = soup.find('div',class_="book-page-tag-section␣relative").find_all('span')
47            for tag in tag_div:
48                tags.append(tag.text)
49
50            desc = soup.find_all('script')[5].text
51            pattern = re.compile(r'Description<\/h4><div␣class=\"trix-content␣mt-3\">(.*?)<\/div>') #
                  regex expression for getting the description text out of the script
52            match = pattern.search(desc)
53            description = match.group(1).strip()
54            description = re.sub('<(.*?)>', '', description)
55
56            img = soup.find('div', class_='book-cover').find('img').get_attribute_list('src')[0]
57
58        except Exception as e:
59            print(e)
60            return None
61
62        reviews_url = f'{url}/community_reviews'
63
64        # request for the COMMUNITY REVIEWS of a certain book
65        req = Request(
66            url=reviews_url,
67            headers={"authority": "www.google.com",
68            "accept": "text/html,application/xhtml+xml,application/xml",
```

```
        "accept-language": "en-US,en;q=0.9",
        "referer": "www.google.com",
        "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/129.0.0.0 Safari/537.36"}
    )

    try:
        response = urlopen(req)
        time.sleep(0.1)
        print(response.status, response.reason)

        webpage = response.read()
        soup_two = bs(webpage, 'html.parser')

        # parsing number of reviews
        n_reviews = soup_two.find('span', class_='text-sm font-medium mt-0.5 ml-[5px] text-darkerGrey dark:text-darkGrey').find('a').text.split(' ')[0]

        # parse tag percentages
        tag_percent = {}
        div_tags_percent = soup_two.find('div', class_='mt-2 moods-list-reviews text-sm flex flex-wrap gap-x-4 gap-y-1.5').find_all('p')
        for tag in div_tags_percent:
            # print(tag)
            l = tag.text.split(' ')
            tag_percent[l[0][:-1]] = l[1][:-1]

        # parse mood percentages
        pace_percent = {}
        div_pace_percent = soup_two.find('div', class_='w-full').find_all('span', class_ = 'sr-only')
        for pace in div_pace_percent:
            l = pace.text.split(' ')
            pace_percent[l[4]] = l[0][:-1]

    except Exception as e:
        print(e)
        return None

    data = {
        'storygraph_id': id,
        'title': title,
        'authors': authors,
        'ISBN/UID': book_id,
        'pages': pages,
        'first_pub': first_pub,
        'tags': tags,
        'description': description,
        'cover-source': img,
        'n_reviews': n_reviews,
        'tag_percent': tag_percent,
        'pace_percent': pace_percent,
    }

    return data
```

Código A.3: Código de la función `thesaurus` del fichero **dictionary_scraper.py**.

```
def thesaurus(word, include='strongest'):

    assert include in ['weak', 'strong', 'strongest'], '\'include\' parameter must have one of these values: ['weak', 'strong', 'strongest']'

    word = re.sub(" ", "%20", word) # useful parsing for urls
    url = f'https://www.thesaurus.com/browse/{word}?noredirect=true'

    # request for the webpage of a certain book
    req = Request(
        url=url,
        headers={"authority": "www.google.com",
        "accept": "text/html,application/xhtml+xml,application/xml",
        "accept-language": "en-US,en;q=0.9",
        "referer": "www.google.com",
        "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/129.0.0.0 Safari/537.36"}
    )

    response = urlopen(req)
    soup = bs(response.read(), 'html.parser')
    synonyms = soup.find('section', class_="synonym-antonym-panel")
    s_list = []

    # always include strongest synonyms
    strongest = synonyms.find_all('a', class_="word-chip synonym-antonym-word-chip similarity-100")
    for a in strongest:
        parsed_text = re.sub(r"  +", "", a.text)
        parsed_text = re.sub("\n", "", parsed_text)
        s_list.append(parsed_text)

    if include == 'strong':
        strong = synonyms.find_all('a', class_="word-chip synonym-antonym-word-chip similarity-50")
        for a in strong:
            parsed_text = re.sub(r"  +", "", a.text)
            parsed_text = re.sub("\n", "", parsed_text)
            s_list.append(parsed_text)

    if include == 'weak':
        strong = synonyms.find_all('a', class_="word-chip synonym-antonym-word-chip similarity-50")
        for a in strong:
            parsed_text = re.sub(r"  +", "", a.text)
            parsed_text = re.sub("\n", "", parsed_text)
            s_list.append(parsed_text)

        weak = synonyms.find_all('a', class_="word-chip synonym-antonym-word-chip similarity-10")
        for a in weak:
            parsed_text = re.sub(r"  +", "", a.text)
            parsed_text = re.sub("\n", "", parsed_text)
            s_list.append(parsed_text)

    return [s_list, len(s_list)]
```

Código A.4: Código de la función `get_lastfm_music` del fichero **lastfm_scraper.py**

```
1  def get_lastfm_music(tag_list, network, n):
2
3      final_dict = {}
4      a_count = 0
5      a_total = len(tag_list)
6      print(f"Processing␣{a_total}␣tags")
7
8      for kw in tag_list:
9          final_dict[kw] = {'TRACK': {'n': 0, 'items': []}, 'FROM_ARTISTS': {'n': 0, 'items': []},
               'FROM_ALBUMS': {'n': 0, 'items': []}}
10         a_count += 1
11         print(f"{kw}␣(tag␣{a_count}␣of␣{a_total})")
12
13         try:
14             tracks = [(tr.item.get_name(), tr.item.get_artist().get_name(), kw) for tr in
                   network.get_tag(kw).get_top_tracks(limit=10*n)]
15             print(f"Got␣{len(tracks)}␣tracks")
16             final_dict[kw]['TRACK']['n'] += len(tracks)
17             final_dict[kw]['TRACK']['items'] += tracks
18         except Exception as e:
19             print(e)
20             print(f"Something␣went␣wrong␣with␣tracks␣of␣{kw}")
21
22         try:
23             albums = network.get_tag(kw).get_top_albums(limit=n)
24             a_tracks = []
25             print(f"Got␣{len(albums)}␣albums")
26             counter = 0
27             for album in albums:
28                 counter += 1
29                 if counter < len(albums):
30                     print(f"\rGathering␣album␣{counter}", end = "")
31                 else:
32                     print(f"\rGathering␣album␣{counter}")
33                 a_tracks += [(tr.get_name(), tr.get_artist().get_name(), kw) for tr in
                       album.item.get_tracks()]
34             final_dict[kw]['FROM_ALBUMS']['n'] += len(a_tracks)
35             final_dict[kw]['FROM_ALBUMS']['items'] += a_tracks
36         except Exception as e:
37             print(e)
38             print(f"Something␣went␣wrong␣with␣albums␣of␣{kw}")
```

```
39            try:
40                artists = network.get_tag(kw).get_top_artists(limit=n)
41                ar_tracks = []
42                print(f"Got␣{len(artists)}␣artists")
43                counter = 0
44                for artist in artists:
45                    counter += 1
46                    ar_tracks += [(tr.item.get_name(), tr.item.get_artist().get_name(), kw) for tr in
                          artist.item.get_top_tracks(limit=10)]
47                    if counter < len(artists):
48                        print(f"\rGathering␣artist␣{counter}", end = "")
49                    else:
50                        print(f"\rGathering␣artist␣{counter}")
51                final_dict[kw]['FROM_ARTISTS']['n'] += len(ar_tracks)
52                final_dict[kw]['FROM_ARTISTS']['items'] += ar_tracks
53            except Exception as e:
54                print(e)
55                print(f"Something␣went␣wrong␣with␣artists␣of␣{kw}")
56
57        return final_dict
```

Código A.5: Código de la función `spotify_search` del fichero **spotify_filter.py**.

```
def spotify_search(search_url, track, artist, kw, headers, time_to_sleep):

    match_found = False
    rate_limit = False

    print("--------------------")
    print(f"({track}, {artist}, {kw})")
    parsed_track = re.sub(' ', '%20', track)
    parsed_artist = re.sub(' ', '%20', artist)
    search_query = f'{search_url}?q=[...]'

    try:
        time.sleep(time_to_sleep)
        response = requests.get(search_query, headers=headers)
        print(response.status_code, response.reason)
        if response.status_code == 429:
            rate_limit = True
            print(response.headers)
            timeout = int(response.headers['Retry-After'])
            return {'match_found': match_found,
                    'rate_limit': rate_limit,
                    'timeout': timeout}

        items = response.json()

        for item in items['tracks']['items']:
            track_name = item['name']
            artists = []
            for a in item['artists']:
                artists.append(a['name'])
            if track == track_name:
                if artist in artists:
                    match_found = True
                    spotify_item = item
                    break
    except Exception as e:
        print('An exception ocurred')
        print(e)
        print(response.raw)

    if match_found:
        return {'match_found': match_found,
                'rate_limit': rate_limit,
                'track_item': spotify_item}
    else:
        return {'match_found': match_found,
                'rate_limit': rate_limit}
```

B

Página web

Este capítulo muestra los ficheros relevantes para la construcción de la página web, tanto el front-end como el back-end.

Código B.1: Código del fichero **settings.py**.

```
"""
Django settings for tfg_webpage project.
"""

from pathlib import Path
import os

from dotenv import load_dotenv

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

load_dotenv(BASE_DIR / '.env')

DEBUG = os.getenv('DEBUG') # should be 1
if DEBUG == None:
    DEBUG = True
    SECRET_KEY = 'django-insecure-[...]'
    ALLOWED_HOSTS = ['*']
else:
    SECRET_KEY = os.getenv('SECRET_KEY')
    ALLOWED_HOSTS = os.getenv('ALLOWED_HOSTS').split(' ')

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
```

```
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'webapp.apps.WebappConfig',
    'django_extensions',
    'corsheaders',
    'rest_framework',
]

MIDDLEWARE = [
    "corsheaders.middleware.CorsMiddleware",
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

CORS_ORIGIN_ALLOW_ALL = False

CORS_ALLOWED_ORIGINS = [
    'http://localhost:5173',
    'http://127.0.0.1:5173',
    'http://localhost:3000',
    'http://127.0.0.1:3000',
    'https://tfg-book-to-soundtrack-site.onrender.com'
]

ROOT_URLCONF = 'tfg_webpage.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'tfg_webpage.wsgi.application'

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
```

```
    }
}

# Password validation
# https://docs.djangoproject.com/en/5.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

GRAPH_MODELS = {
  'all_applications': False,
  'app_labels': ["webapp"],
}


# Internationalization
# https://docs.djangoproject.com/en/5.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.2/howto/static-files/

STATIC_URL = 'static/'

# Default primary key field type
# https://docs.djangoproject.com/en/5.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

Código B.2: Código del fichero **urls.py**.

```
"""
URL configuration for tfg webpage project.
"""
from django.contrib import admin
from django.urls import path, include
from rest_framework import routers
from webapp import views

# un router que controle específicamente todos los endpoints de la api
api_router = routers.DefaultRouter()
# En el router vamos añadiendo los endpoints a los viewsets
api_router.register('books', views.BooksDB, basename = 'BooksDB')
api_router.register('storygraph-search', views.StorygraphSearch, basename='StorygraphSearch')
api_router.register('isbn', views.GetBookByISBNorUID, basename='GetBookByISBN')
api_router.register('tag', views.getTagSynonyms, basename='GetTagSynonyms')
api_router.register('track-pool', views.getTrackPool, basename='getTrackPool')

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/v1/', include(api_router.urls))
]
```

Código B.3: Código del fichero **models.py**.

```
from django.db import models


class Storygraph_Book(models.Model):
    storygraph_id = models.CharField(max_length=300, default='')
    title = models.CharField(max_length=300)
    authors = models.JSONField(max_length=500)
    isbn_uid = models.CharField(max_length=300, default='')
    pages = models.IntegerField()
    first_published_year = models.IntegerField()
    description = models.TextField(default='')
    cover_source = models.URLField(default='')
    number_of_reviews = models.CharField(max_length=300, default='')
    tag_percentages = models.JSONField(max_length=3000, default=dict)
    pace_percentages = models.JSONField(max_length=3000, default=dict)
    tag_weights = models.JSONField(max_length=3000, default=dict)

    def __str__(self):
        return f"{self.title} by {self.authors}"


class Storygraph_Tag(models.Model):
    tag_name = models.CharField(max_length=100)
```

```
24
25      # MOOD,GENRE,PACE,OTHER
26      tag_type = models.CharField(max_length=100)
27
28      def __str__(self):
29          return f"{self.tag_name} ({self.tag_type})"
30
31
32  class Tagged_Book(models.Model):
33      book = models.ForeignKey(Storygraph_Book, on_delete=models.CASCADE)
34      tag = models.ForeignKey(Storygraph_Tag, on_delete=models.CASCADE)
35      weight = models.FloatField(default=-1)
36
37      def __str__(self):
38          return f"{self.book.title} tagged with {self.tag.tag_name}"
39
40
41  class Synonym(models.Model):
42      synonym = models.CharField(max_length=100)
43
44      # SELF,THESAURUS
45      source = models.CharField(max_length=100)
46
47      def __str__(self):
48          return f"{self.synonym} ({self.source})"
49
50
51  class Synonym_Relation(models.Model):
52      tag = models.ForeignKey(Storygraph_Tag, on_delete=models.CASCADE)
53      synonym = models.ForeignKey(Synonym, on_delete=models.CASCADE)
54      affinity = models.CharField(max_length=50, default='') # [weak, strong, strongest]
55
56      def __str__(self):
57          return f"{self.synonym.synonym} ({self.synonym.source}) is a synonym of {self.tag.tag_name}
                with affinity {self.affinity}"
58
59
60  class LastFM_Entity(models.Model):
61      name = models.CharField(max_length=100)
62      artist = models.CharField(max_length=100)
63      tag = models.CharField(max_length=100, default='')
64      def __str__(self):
65          return f"{self.name} by {self.artist}"
66
67  class Entity_Tag_Relation(models.Model):
68      entity = models.ForeignKey(LastFM_Entity, on_delete=models.CASCADE)
69      tag = models.ForeignKey(Synonym, on_delete=models.CASCADE)
70      source = models.CharField(max_length=100, default='') # TRACK, FROM_ARTISTS,
            FROM_ALBUMS
71
72      def __str__(self):
73          return f"{self.entity.name} is related to {self.tag.synonym}"
```

```
74  class Spotify_Track(models.Model):
75      name = models.CharField(max_length=100)
76      artist = models.CharField(max_length=100)
77      tag = models.CharField(max_length=100, default='')
78      def __str__(self):
79          return f"{self.name} by {self.artist}"
80      spotify_json = models.JSONField(max_length=15000, default=dict)
81
82  class Spotify_Tag_Relation(models.Model):
83      track = models.ForeignKey(Spotify_Track, on_delete=models.CASCADE)
84      tag = models.ForeignKey(Synonym, on_delete=models.CASCADE)
85      source = models.CharField(max_length=100, default='') # TRACK, FROM_ARTISTS,
            FROM_ALBUMS
86
87      def __str__(self):
88          return f"{self.track.name} is related to {self.tag.synonym}"
```

Código B.4: Código del fichero **get_track_pool.py**.

```
1   def get_track_pool(book, n_tracks):
2   # returns the complete track pool of a certain book, stored in the database as is.
3
4     # calculates the number of tracks per tag (based on total number of tracks)
5     tag_tracks = [(t[0], float(n_tracks)*t[1]) for t in book['tag_weights']]
6     tag_residual = [[t[0], t[1]-math.floor(t[1]), int(t[1]-(t[1]-math.floor(t[1])))] for t in tag_tracks]
7
8     sum_tracks = 0
9     for t in tag_residual:
10      sum_tracks += t[2]
11    dif_tracks = int(n_tracks) -sum_tracks
12
13    if dif_tracks > 0:
14      tag_residual.sort(key=lambda t: t[1], reverse=True)
15      i = 0
16      while dif_tracks > 0:
17        dif_tracks -= 1
18        tag_residual[i][2] += 1
19        i += 1
20
21    sum_tracks = 0
22    for t in tag_residual:
23      sum_tracks += t[2]
24    if sum_tracks == int(n_tracks):
25      print('Check!')
26    else:
27      print('Something is wrong')
28
29    synonyms = {}
30    synonyms_two = {}
31    for t in tag_residual:
```

```
    synonyms[t[0]] = {'n_tracks': t[2], 'synonyms': {}}
    synonyms_two[t[0]] = {'n_tracks': t[2], 'tracks': []}

  for t in synonyms.keys():
    tag_id = Storygraph_Tag.objects.get(tag_name=t).id
    s_list = [Synonym.objects.get(id=sr.synonym_id).synonym for sr in
        Synonym_Relation.objects.filter(tag_id=tag_id)]
    for s in s_list:
      synonyms[t]['synonyms'][s] = []

  for tag in synonyms.keys():
    for syn in synonyms[tag]['synonyms'].keys():
      syn_id = Synonym.objects.get(synonym=syn).id
      list_objects = [Spotify_Track.objects.get(id=rt.track_id) for rt in
          Spotify_Tag_Relation.objects.filter(tag_id=syn_id)]
      tracklist = []
      for t in list_objects:
        if t.spotify_json['album']['images'] == []:
          tracklist.append((t.name, t.artist, t.tag,
              t.spotify_json['uri'],t.spotify_json['external_urls']['spotify']))
        else:
          tracklist.append(((t.name, t.artist, t.tag,
              t.spotify_json['uri'],t.spotify_json['external_urls']['spotify'],
              t.spotify_json['album']['images'][0]['url'])))
      synonyms[tag]['synonyms'][syn] += tracklist
      synonyms_two[tag]['tracks'] += tracklist

  #### useful for debug ######
  n_tracks = 0
  for tag in synonyms.keys():
    print(tag)
    print(len(synonyms[tag]['synonyms'].items()))
    n_by_syn = 0
    for syn in synonyms[tag]['synonyms'].keys():
      n_by_syn += len(synonyms[tag]['synonyms'][syn])
      n_tracks += len(synonyms[tag]['synonyms'][syn])
    print(n_by_syn)

  return([n_tracks, synonyms_two, synonyms])
```

Código B.5: Código del fichero **views.py**.

```
import json
from django.core import serializers
from .models import Storygraph_Book, Storygraph_Tag, Synonym, Synonym_Relation
from .serializers import BookDBSerializer
from rest_framework import mixins
from rest_framework import generics
from rest_framework import viewsets
from .scripts.storygraph_scraper import book_search, book_info, add_weights_to_book
from .scripts.get_track_pool import get_track_pool
from rest_framework.response import Response
```

```
11
12  class BooksDB(viewsets.ReadOnlyModelViewSet):
13  # supports list and retrieve pk
14      queryset = Storygraph_Book.objects.all().order_by('authors').values()
15      serializer_class = BookDBSerializer
16
17  class GetBookByISBNorUID(viewsets.ViewSet):
18  # has online and offline mode
19      def create(self, request):
20          # get request with isbn/uid param
21          b = json.loads(request.body.decode("utf-8"))
22          print(b)
23          if b['mode'] == 'offline':
24              book = Storygraph_Book.objects.filter(isbn_uid=b['book_id'])
25              book = serializers.serialize('json', book)
26              book = json.loads(book)
27              return Response(book[0]['fields'])
28          elif b['mode'] == 'online':
29              id = book_search([b['book_id']])
30              book = book_info(id)
31              book = add_weights_to_book(book)
32              return Response(book)
33
34
35  class StorygraphSearch(viewsets.ViewSet):
36  # accesses StoryGraph to retrieve a certain book
37      def create(self, request):
38          # post request with search query in body
39          b = json.loads(request.body.decode("utf-8"))
40          id = book_search([b['searchItem']])
41          book = book_info(id)
42          if book == None:
43              return Response('error')
44          book = add_weights_to_book(book)
45          return Response(book)
46
47  class getTagSynonyms(viewsets.ViewSet):
48  # retrieve the list of synonyms associated to a StoryGraph tag
49      def retrieve(self, request, pk=None):
50          tag = Storygraph_Tag.objects.get(tag_name=pk)
51          query_list = Synonym_Relation.objects.filter(tag_id=tag.id)
52          syn_list = []
53          for s in query_list:
54          syn_list.append((Synonym.objects.get(id=s.synonym_id).synonym, s.affinity)) # (synonym,
                affinity)
55          syn_dict = {pk:{'strongest':[], 'strong': [], 'weak': []}}
56          for s in syn_list:
57              syn_dict[pk][s[1]].append(s[0])
58
59          return Response(syn_dict)
```

```
class getTrackPool(viewsets.ViewSet):
# has offline and online mode
    def create(self, request):
        # receives the book, returns the track pool
        b = json.loads(request.body.decode("utf-8"))

        # gather the book either offline (from db) or online (from scraping)
        if b['mode'] == 'offline':
            book = Storygraph_Book.objects.filter(isbn_uid=b['book_id'])
            book = serializers.serialize('json', book)
            book = json.loads(book)
            book = book[0]['fields']
        elif b['mode'] == 'online':
            id = book_search([b['book_id']])
            book = book_info(id)
            book = add_weights_to_book(book)

        return Response(get_track_pool(book, b['n_tracks']))
```

Código B.6: Código del fichero **App.jsx**.

```
import { BrowserRouter, Routes, Route } from "react-router-dom"
import InstructionsView from "./views/instructions/instructionsView"
import MainFlowView from './views/mainFlow/mainFlowView'
import ChooseBookView from "./views/chooseBook/chooseBookView"
import SelectedBookView from "./views/selectedBook/selectedBookView"
import ConfigurePlaylistView from "./views/configurePlaylist/configurePlaylistView"
import ShowPlaylistView from "./views/showPlaylist/showPlaylistView"
import './App.css'

function App() {
  return (
        <BrowserRouter>
            <Routes>
                <Route path="/" element={<MainFlowView />} />
                <Route path="/instructions" element={<InstructionsView />} />
                <Route path="/flow/choose-book" element={<ChooseBookView />} />
                <Route path="/flow/book/:mode/:book_id" element={<SelectedBookView />} />
                <Route path="/flow/book/:mode/:book_id/create-playlist"
                    element={<ConfigurePlaylistView />} />
                <Route path="/flow/book/:mode/:book_id/show-playlist/:n_tracks"
                    element={<ShowPlaylistView />} />
                <Route path="*" element={<h1>Not found!</h1>} />
            </Routes>
        </BrowserRouter>
    );
}

export default App
```

Código B.7: Fichero **requirements.txt**.

```
beautifulsoup4==4.14.3
Django==6.0.4
djangorestframework==3.17.0
django-extensions==4.1
django-cors-headers==4.9.0
pylast==7.0.2
pymultidictionary==1.3.2
python-dotenv==1.2.2
Requests==2.33.1
python-dotenv==1.2.2
gunicorn==21.2.0
```

Código B.8: Fichero **build_production.sh**.

```
#!/usr/bin/env bash

set -o errexit # exit on error

pip install -r requirements.txt

# python manage.py collectstatic --no-input
python manage.py migrate

python manage.py createsu # already created tho
```

Código B.9: Fichero **package.json**.

```
{
  "name": "webapp-frontend-react",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint .",
    "preview": "vite preview --port 5173"
  },
  "dependencies": {
    "@mui/material": "^7.3.9",
    "react": "^19.2.4",
    "react-dom": "^19.2.4",
    "react-router-dom": "^7.13.2"
  },
  "devDependencies": {
    "@eslint/js": "^9.39.4",
    "@types/react": "^19.2.14",
    "@types/react-dom": "^19.2.3",
    "@vitejs/plugin-react": "^6.0.1",
    "eslint": "^9.39.4",
    "eslint-plugin-react-hooks": "^7.0.1",
    "eslint-plugin-react-refresh": "^0.5.2",
    "globals": "^17.4.0",
    "vite": "^8.0.1"
  }
}
```

UÁM
Universidad Autónoma
de Madrid