

PROYECTO FINAL DE CICLO

C.F.G.S DESARROLLO
DE APLICACIONES WEB

CANCIONERO DIGITAL SALESIANO

Jerome Gamboa y Marco Batista

COLEGIO NUESTRA SEÑORA DEL PILAR

TÍTULO	Cancionero digital salesiano interactivo.
AUTOR	Jerome Gamboa y Marco Batista.
TUTOR	Rosa Rodríguez.
FECHA	18 de Junio de 2025
CICLO	C.F.G.S Desarrollo de Aplicaciones Web.

RESUMEN DEL PROYECTO

Este Trabajo de Fin de Grado presenta el diseño, desarrollo e implementación de una aplicación web centrada en la gestión y visualización de canciones litúrgicas con acordes, orientada a entornos de celebración religiosa y comunitaria, especialmente en el ámbito salesiano. El objetivo principal del proyecto es ofrecer una herramienta accesible y dinámica que facilite la organización de repertorios y la participación activa mediante funcionalidades interactivas, como el cambio de tonalidad, la categorización litúrgica o la personalización de listas de canciones.

El desarrollo se realiza íntegramente con el framework Django, utilizando su sistema de plantillas, ORM y capacidades de gestión de usuarios, junto con HTML, CSS y JavaScript para el frontend. Entre las funcionalidades implementadas destacan: la visualización adaptativa de canciones según el tiempo litúrgico, la transposición de acordes en tiempo real, la creación de listas personalizadas por usuario y una interfaz diseñada para facilitar la lectura y la participación comunitaria.

La metodología empleada es principalmente incremental, siguiendo principios de desarrollo ágil. Se utilizan herramientas como Git para el control de versiones, Django Allauth para la autenticación y autorización, y AJAX para mejorar la experiencia de usuario mediante interacciones sin recarga de página. Cada fase del proyecto se documenta cuidadosamente, prestando especial atención a la claridad del código y su mantenibilidad futura.

Como parte del desarrollo, se abordan aspectos específicos como la simplificación de la notación musical (incluyendo la adaptación de sostenidos y bemoles a sus equivalentes enarmónicos), la normalización de cadenas con caracteres Unicode, el cálculo de fechas móviles como la Pascua y su relación con la clasificación litúrgica, y la posibilidad de trabajar con archivos JSON para la carga y exportación de contenidos.

Entre las funcionalidades planteadas, pero aún no desarrolladas, completamente se encuentran: el almacenamiento personalizado de preferencias de tonalidad por usuario, la incorporación de un modo oscuro, la traducción de acordes a notación inglesa y un sistema comunitario para la subida y publicación de versiones cantadas de las canciones, lo cual se propone como línea futura de mejora.

Los resultados obtenidos muestran una aplicación funcional, estable y preparada para su uso real, con un modelo de datos escalable y una arquitectura que permite seguir añadiendo mejoras. Las pruebas realizadas confirman un rendimiento correcto en diferentes dispositivos y navegadores, así como una aceptación positiva por parte de los primeros usuarios de prueba.

En conclusión, el proyecto no solo cumple con los objetivos técnicos establecidos, sino que también aporta valor a nivel pastoral y comunitario, fomentando la participación activa en las celebraciones mediante el uso de tecnología accesible y personalizada.

ÍNDICE

I. DESCRIPCIÓN Y ALCANCE	1
1. Descripción general del proyecto	
2. Objetivos	
3. Tecnología utilizada	
II. PLANIFICACIÓN	2
1. Planificación inicial	
2. Planificación real	
III. ANÁLISIS, DISEÑO E IMPLEMENTACIÓN	3
1. Análisis de requerimientos: catálogo de requisitos	
a) Requisitos funcionales	
b) Requisitos no funcionales	
2. Diseño del sistema: arquitectura	
a) Hardware	
b) Software	
c) Restricciones	
3. Diseño del sistema	
a) Diagramas ER de bases de datos(opcional)	
b) Tablas(opcional)	
c) Diagramas de clases (si procede)	
d) Diseño de la interfaz de usuario	
e) Implementaciones (opcional)	

IV. PLAN DE PRUEBAS	4
V. CONCLUSIÓN	5
VI. BIBLIOGRAFÍA	6
VII. GLOSARIO DE TÉRMINOS	7
VIII. ANEXOS	8

I. DESCRIPCIÓN Y ALCANCE DEL PROYECTO

1. Descripción general del proyecto

Justificación y origen del proyecto

El presente proyecto surge a raíz de la necesidad identificada en un **Centro Juvenil Salesiano**, donde participamos activamente como animadores de jóvenes en la organización de celebraciones cantadas, tales como oraciones y eucaristías. Durante años, hemos contado con una aplicación desarrollada en **Java y Android** por uno de nuestros compañeros, la cual ha facilitado la gestión y acceso a los cantos utilizados en estos eventos.

Sin embargo, con el tiempo, la necesidad ha evolucionado, ya que una gran parte del equipo de animadores y participantes emplea **dispositivos iOS (iPhone, iPad)**, lo que ha generado dificultades de acceso a la aplicación existente. Por ello, consideramos fundamental el desarrollo de una **plataforma web** que no solo replique las funcionalidades previamente disponibles, sino que las amplíe y mejore, asegurando así su accesibilidad desde cualquier dispositivo con conexión a Internet.

Análisis del mercado y propuesta de valor

Existen diversas aplicaciones y plataformas que ofrecen recopilaciones de cantos religiosos. Sin embargo, la mayoría están limitadas a formatos estáticos (PDFs, blogs) o requieren suscripciones para su uso completo. Nuestra propuesta se diferencia en los siguientes aspectos:

- **Accesibilidad multiplataforma:** al tratarse de una aplicación web, será accesible desde cualquier navegador, sin importar el sistema operativo.
- **Personalización:** los usuarios podrán **crear listas personalizadas, guardar canciones favoritas y ajustar la tonalidad** de los acordes según sus necesidades.
- **Interactividad y optimización para el ensayo:** incluye funciones como **modo ensayo con scroll automático, cambio de notación de acordes o tipografía adaptable**.
- **Gestión centralizada:** un **panel de administración** permite modificar y añadir nuevas canciones de manera sencilla.

2. Objetivos

Objetivo general

Desarrollar una **plataforma web interactiva** que permita la consulta, organización y reproducción de cantos religiosos utilizados en celebraciones, asegurando su accesibilidad desde cualquier dispositivo y mejorando la experiencia de uso mediante funcionalidades avanzadas.

Objetivos específicos

1. **Desarrollar una interfaz web intuitiva** que facilite la navegación y el acceso a las canciones.
2. **Implementar un sistema de búsqueda eficiente**, permitiendo localizar canciones por título, artista, fragmentos de letra o festividad.
3. **Permitir la personalización de la experiencia del usuario** mediante listas de reproducción, favoritos y ajustes de la tonalidad de los acordes.
4. **Optimizar la lectura y uso de las canciones en ensayos y celebraciones**, incorporando un **modo ensayo con scroll automático** y la posibilidad de cambiar la notación de los acordes.
5. **Garantizar la accesibilidad y adaptabilidad** mediante un **modo oscuro y opciones de ajuste tipográfico**.
6. **Integrar autenticación de usuarios con Google OAuth** para permitir la sincronización de preferencias y listas personalizadas.
7. **Desarrollar un panel de administración** que facilite la edición y adición de nuevas canciones a la plataforma.
8. **Diseñar y optimizar una base de datos en PostgreSQL**, permitiendo una gestión eficiente del contenido.
9. **Realizar pruebas de uso** con usuarios del centro juvenil para mejorar la experiencia final antes del despliegue.

Público objetivo

La aplicación está diseñada principalmente para:

- **Animadores y músicos del Centro Juvenil Salesiano**, quienes necesitan gestionar el repertorio de cantos para oraciones y eucaristías.
- **Comunidades religiosas y grupos parroquiales** que buscan una herramienta accesible y funcional para organizar su música.
- **Cualquier persona interesada en la música litúrgica**, que desee explorar, aprender o utilizar estos cantos en sus propias celebraciones.

3. Tecnología utilizada

Arquitectura general del proyecto

La aplicación se ha desarrollado siguiendo una arquitectura cliente-servidor basada en Django como backend y HTML/CSS/JavaScript en el frontend. La lógica de negocio y gestión de datos reside en el backend, mientras que las interacciones dinámicas y visuales del usuario se gestionan desde el navegador. Se utiliza PostgreSQL como sistema gestor de base de datos relacional.

Frontend

- **Tecnologías:** HTML5, CSS3, JavaScript (vanilla), Bootstrap 5.
- **Interactividad:** Uso de fetch() para peticiones AJAX, botones dinámicos, modales, scroll automático y control de visualización (zoom, ocultar acordes, etc.).
- **Estilización:** Bootstrap 5 junto con django-crispy-forms y el paquete crispy-bootstrap5.

Backend

- **Lenguaje:** Python 3.11.
- **Framework:** Django.
- **Base de datos:** PostgreSQL.
- **Autenticación:** django-allauth, personalizada mediante edición de plantillas y vistas.
- **Módulos personalizados:**
 - Cálculo del tiempo litúrgico actual.
 - Importación de JSON a base de datos.
 - Unificación de tiempos litúrgicos duplicados.
 - Exportación de listas a Word (python-docx).
 - Transposición de acordes dinámicamente.

Entorno de desarrollo

- **Entorno virtual:** Uso de venv para aislamiento de dependencias.
- **Visual Studio Code**
- **Control de versiones:** Git y GitHub, con estrategia basada en ramas y pull requests.
- **Dependencias:** Registradas en requirements.txt para facilitar instalación y despliegue.

Gestión de versiones y colaboración

- Trabajo colaborativo con ramas independientes por funcionalidad.
- Uso de GitHub Desktop para facilitar la interacción con el repositorio.
- Se implementó .gitignore para excluir archivos irrelevantes (como caché de Python).

Consideraciones técnicas destacadas

- **AJAX y mejora UX:** Búsqueda en tiempo real, favoritos, listas personalizadas y transposición de acordes se realizan sin recargar la página.
- **Panel de administración personalizado:** Mejora la gestión de datos del proyecto.
- **Exportación Word:** Generación dinámica de documentos .docx con o sin acordes.
- **Seguridad:** Todas las vistas que gestionan datos están protegidas con @login_required.

II. PLANIFICACIÓN

1. Planificación inicial

a) Planificación General

Descripción de la Fase	Fecha de inicio	Fecha de finalización	Duración (días)	Horas estimadas
Plan de trabajo	10/03/2025	10/03/2025	1	3 h
Análisis y diseño	11/03/2025	14/03/2025	4	16 h
Implementación	13/04/2025	26/04/2025	14	85 h
Despliegue	28/04/2025	30/04/2025	3	12 h
Pruebas y optimización	01/05/2025	08/05/2025	8	12 h
Memoria y presentación	09/05/2025	09/06/2025	31	20 h

Nota: Algunas tareas pueden desarrollarse en paralelo, permitiendo optimizar el tiempo de trabajo.

b) Plan de Trabajo Detallado

1. Análisis y Diseño (16 h)

Tarea	Fecha de inicio	Fecha de finalización	Duración (días)	Horas estimadas
Definir descripción, objetivos y análisis de requisitos.	11/03/2025	11/03/2025	1	4 h
Creación de marca	11/03/2025	11/03/2025	1	1h
Definir estructura de la base de datos (PostgreSQL)	12/03/2025	12/03/2025	1	4 h
Diseño UI/UX (pantallas y navegación)	13/03/2025	13/03/2025	1	4 h
Organización de GitHub y de equipo de trabajo (Instalación versión Django)	14/03/2025	14/03/2025	1	3 h

2. Implementación (85 h)

2.1 Pantalla Inicial (7 h)

Tarea	Fecha de inicio	Fecha de finalización	Horas estimadas
Implementación del banner principal	07/04/2025	09/04/2025	3 h
Mostrar las canciones por tiempo litúrgico	07/04/2025	09/04/2025	3 h
Buscador de canciones (título, artista, etc.)	10/04/2025	10/04/2025	2 h

2.2 Sidepanel - Menú Lateral (10 h)

Tarea	Fecha de inicio	Fecha de finalización	Horas estimadas
Implementación del menú lateral	07/04/2025	09/04/2025	2 h
Sección "Favoritos"	11/04/2025	11/04/2025	3 h
Listas personalizadas	12/04/2025	13/04/2025	5 h

2.3 Funcionalidades dentro de una canción (30 h)

Tarea	Fecha de inicio	Fecha de finalización	Horas estimadas
Implementar cambio de tonalidad	14/04/2025	15/04/2025	6 h
Cambio de notación de acordes	16/04/2025	16/04/2025	4 h
Cambio de zoom	16/04/2025	16/04/2025	1 h
Añadir a lista personalizada	18/04/2025	18/04/2025	3 h
Crear/seleccionar listas	19/04/2025	19/04/2025	3 h
Exportar lista a Docs/PDF/Markdown	20/04/2025	20/04/2025	4 h
Mover/eliminar canciones en listas	21/04/2025	21/04/2025	4 h
Modo Ensayo (scroll automático)	22/04/2025	23/04/2025	5 h

2.4 Configuración Avanzada (7 h)

Tarea	Fecha de inicio	Fecha de finalización	Horas estimadas
Implementar modo oscuro	24/04/2025	24/04/2025	4 h
Tipografía adaptable	25/04/2025	25/04/2025	3 h

2.5 Base de Datos (10 h)

Tarea	Fecha de inicio	Fecha de finalización	Horas estimadas
Serialización de JSON a SQL en PostgreSQL	31/03/2025	02/04/2025	4 h
Investigación e Implementación Google OAuth para usuarios	03/04/2025	04/04/2025	6 h

2.6 Panel de Administración (13 h)

Tarea	Fecha de inicio	Fecha de finalización	Horas estimadas
Creación del panel de administración	07/04/2025	09/04/2025	8 h
Funcionalidad para editar y crear canciones	10/04/2025	11/04/2025	5 h

3. Pruebas y Optimización (12 h)

Tarea	Fecha de inicio	Fecha de finalización	Horas estimadas
Pruebas con usuarios y feedback	05/05/2025	06/05/2025	8 h
Corrección de errores	07/05/2025	08/05/2025	4 h

4. Despliegue

Tarea	Fecha de inicio	Fecha de finalización	Duración (días)	Horas estimadas
Investigación e implementación	28/04/2025	30/04/2025	3	12 h

5. Memoria y Presentación (20 h)

Tarea	Fecha de inicio	Fecha de finalización	Horas estimadas
Documentación técnica del proyecto	09/05/2025	16/05/2025	6 h
Elaboración de la memoria	17/05/2025	01/06/2025	10 h
Preparación de la presentación final	02/06/2025	09/06/2025	4 h

Método de Trabajo y Seguimiento

Para gestionar el trabajo de manera eficiente, se utiliza **Trello**, mediante la organización de las tareas en columnas según su estado de avance:

1. **Backlog:** Tareas pendientes.
2. **En progreso:** Tareas en desarrollo.
3. **En revisión:** Tareas finalizadas pendientes de validación.
4. **Hecho:** Tareas completadas.

Cada semana se asignan tareas y se revisa el progreso. Se prioriza la comunicación asincrónica mediante Trello y Discord/WhatsApp para coordinar el trabajo.

2. Planificación real

Nota:

- Se utiliza el color de fondo verde para indicar que la planificación inicial ha sido modificada.
- Algunas tareas pueden desarrollarse en paralelo.

c) Planificación General

Descripción de la Fase	Fecha de inicio	Fecha de finalización	Duración (días)	Horas estimadas
Plan de trabajo	10/03/2025	10/03/2025	1	3 h
Análisis y diseño	11/03/2025	31/03/2025	21	23 h
Implementación	01/04/2025	02/06/2025	63	94 h
Despliegue	03/06/2025	04/06/2025	2	12 h
Pruebas y optimización	05/06/2025	07/06/2025	3	12 h
Memoria y presentación	08/06/2025	10/06/2025	3	64 h

d) Plan de Trabajo Detallado

1. Análisis y Diseño (23h)

Tarea	Fecha de inicio	Fecha de finalización	Duración (días)	Horas estimadas
Definir descripción, objetivos y análisis de requisitos.	11/03/2025	11/03/2025	1	4 h
Creación de marca	11/03/2025	11/03/2025	1	1h
Definir estructura de la base de datos (PostgreSQL)	12/03/2025	31/03/2025	2	8 h
Diseño UI/UX (pantallas y navegación)	13/03/2025	31/03/2025	2	6h
Organización de GitHub y de equipo de trabajo (Instalación versión Django)	13/03/2025	13/03/2025	1	4 h

2. Implementación (94 h)

2.1 Pantalla Inicial (8 h)

Tarea	Fecha de inicio	Fecha de finalización	Horas estimadas
Implementación del banner principal	07/04/2025	11/04/2025	3 h
Mostrar las canciones por tiempo litúrgico	07/04/2025	11/04/2025	3 h
Buscador de canciones (título, artista, etc.)	07/04/2025	11/04/2025	2 h

2.2 Sidepanel - Menú Lateral (18 h)

Tarea	Fecha de inicio	Fecha de finalización	Horas estimadas
Implementación del menú lateral	07/04/2025	11/04/2025	2 h
Sección "Favoritos"	12/04/2025	16/04/2025	6 h
Listas personalizadas	17/04/2025	21/04/2025	10 h

2.3 Funcionalidades dentro de una canción (29 h)

Tarea	Fecha de inicio	Fecha de finalización	Horas estimadas
Implementar cambio de tonalidad	22/04/2025	24/04/2025	6 h
Cambio de notación de acordes	X	X	X
Cambio de zoom	27/04/2025	27/04/2025	1 h
Añadir a lista personalizada	28/04/2025	28/04/2025	3 h
Crear/seleccionar listas	29/04/2025	29/04/2025	3 h
Exportar lista a Docs/PDF/Markdown	30/04/2025	30/04/2025	4 h
Mover/eliminar canciones en listas	01/05/2025	01/05/2025	4 h
Modo Ensayo (scroll automático)	02/05/2025	04/05/2025	5 h

2.4 Configuración Avanzada (7 h)

Tarea	Fecha de inicio	Fecha de finalización	Horas estimadas
Implementar modo oscuro	05/05/2025	06/05/2025	4 h
Tipografía adaptable	07/05/2025	07/05/2025	3 h

2.5 Base de Datos (20 h)

Tarea	Fecha de inicio	Fecha de finalización	Horas estimadas
Serialización de JSON a SQL en PostgreSQL	31/03/2025	02/04/2025	4 h
Investigación e Implementación Google OAuth para usuarios	08/05/2025	09/05/2025	16 h

2.6 Panel de Administración (12 h)

Tarea	Fecha de inicio	Fecha de finalización	Horas estimadas
Creación del panel de administración	07/04/2025	07/04/2025	2 h
Funcionalidad para editar y crear canciones	10/05/2025	13/05/2025	10 h

3. Pruebas y Optimización (12 h)

Tarea	Fecha de inicio	Fecha de finalización	Horas estimadas
Pruebas con usuarios y feedback	06/06/2025	07/06/2025	8 h
Corrección de errores	07/06/2025	07/06/2025	4 h

4. Despliegue

Tarea	Fecha de inicio	Fecha de finalización	Horas estimadas
Investigación e implementación	03/06/2025	04/06/2025	12 h

5. Memoria y Presentación (64 h)

Tarea	Fecha de inicio	Fecha de finalización	Horas estimadas
Documentación técnica del proyecto	01/05/2025	08/05/2025	20 h
Elaboración de la memoria	01/05/2025	08/06/2025	40 h
Preparación de la presentación final	09/06/2025	10/06/2025	4 h

III. ANÁLISIS, DISEÑO E IMPLEMENTACIÓN

1. Análisis de requerimientos: catálogo de requisitos

a) Requisitos funcionales

Los requisitos funcionales describen las funcionalidades que debe ofrecer la aplicación para cumplir con sus objetivos. Se dividen en diferentes módulos o subsistemas.

1.1. Gestión de Canciones

- La aplicación debe permitir la búsqueda de canciones por **título, artista, fragmentos de letra o festividad**.
- Las canciones deben mostrar **letra y acordes** correctamente formateados.
- Se debe permitir el **cambio de tonalidad** en los acordes.
- Se debe permitir el **cambio de notación de acordes (americana ↔ europea)**.
- Se debe permitir **ajustar el tamaño de la letra (zoom)** para mejorar la accesibilidad.
- Implementar un **modo ensayo**, que permita hacer un **scroll automático** de la letra a una velocidad ajustable.

1.2. Gestión de Usuarios

- Los usuarios deben poder **autenticarse mediante Google OAuth**.
- Un usuario autenticado debe poder **guardar sus canciones favoritas**.
- Los usuarios deben poder **crear y gestionar listas personalizadas de canciones**.
- Debe guardarse la **última tonalidad usada por cada usuario** en cada canción.

1.3. Menú de Navegación y Panel Lateral

- La aplicación debe contar con un **menú lateral** con accesos directos a:
 - Pantalla de inicio
 - Favoritos
 - Listas personalizadas
 - Configuración
- Las listas personalizadas deben permitir:
 - **Añadir canciones**.
 - **Reordenar el orden de las canciones** dentro de la lista.
 - **Eliminar canciones de la lista**.
 - **Exportar la lista en PDF, Markdown o Google Docs** sin incluir los acordes.

1.4. Configuración Avanzada

- Se debe implementar un **modo oscuro** para mejorar la experiencia visual.
- La aplicación debe permitir **ajustar la tipografía** para mejorar la accesibilidad.

1.5. Administración del Contenido

- Se debe desarrollar un **panel de administración** accesible solo para usuarios con permisos especiales.
- Los administradores podrán **añadir nuevas canciones y modificar las existentes**.

1.6. Base de Datos y Persistencia

- Se debe almacenar la información en una **base de datos PostgreSQL**.
- Se debe permitir **importar un archivo JSON con las canciones** al sistema.
- La base de datos debe permitir la gestión de usuarios autenticados y sus preferencias.

1.7. Pruebas y Despliegue

- Se deben realizar **pruebas funcionales** en cada módulo de la aplicación.
- Se debe implementar un **sistema de pruebas con usuarios reales** para mejorar el uso.
- Se debe optimizar la aplicación para garantizar **un buen rendimiento en distintos dispositivos y navegadores**.

b) Requisitos no funcionales

Los requisitos no funcionales describen las características generales del sistema que afectan su calidad y rendimiento, pero no son directamente funcionalidades implementadas.

2.1. Accesibilidad y Uso

- La aplicación debe ser **intuitiva y fácil de usar** para usuarios sin conocimientos técnicos.
- Debe estar optimizada para **uso en dispositivos móviles y de escritorio**.
- Se debe garantizar la **compatibilidad con navegadores modernos** (Chrome, Firefox, Safari, Edge).

2.2. Seguridad y Autenticación

- La autenticación de usuarios debe realizarse mediante **Google OAuth** para evitar el manejo de contraseñas.
- Los datos de usuarios (favoritos, listas) deben estar correctamente **protegidos y almacenados de forma segura**.
- Se deben implementar **políticas de acceso restringido** en el panel de administración.

2.3. Rendimiento y Escalabilidad

- La aplicación debe ser capaz de **cargar y mostrar cientos de canciones sin pérdida de rendimiento**.
- Se debe optimizar el uso de la base de datos para evitar **consultas innecesarias** y mejorar la velocidad.
- El backend debe ser escalable para **soportar un crecimiento en el número de usuarios**.

2.4. Mantenimiento y Actualización

- Se debe garantizar que la aplicación pueda ser **actualizada fácilmente sin afectar a los usuarios**.
- La base de datos debe permitir la **modificación y adición de nuevas canciones** de manera sencilla.
- El código debe ser **modular y bien documentado** para facilitar su mantenimiento.

2. Diseño del sistema: arquitectura

e) Hardware

Para el funcionamiento del sistema no se requiere equipamiento especializado. Tanto el servidor como los clientes pueden operar con hardware actual de gama media:

- **Servidor:** puede ser físico o virtualizado, con recursos estándar (procesador moderno de varios núcleos, al menos 8 GB de RAM y almacenamiento SSD).
- **Cliente:** cualquier ordenador personal de uso común, con características equivalentes a un equipo de gama media actual.

f) Software

El sistema funciona correctamente en entornos estándar. A nivel de software:

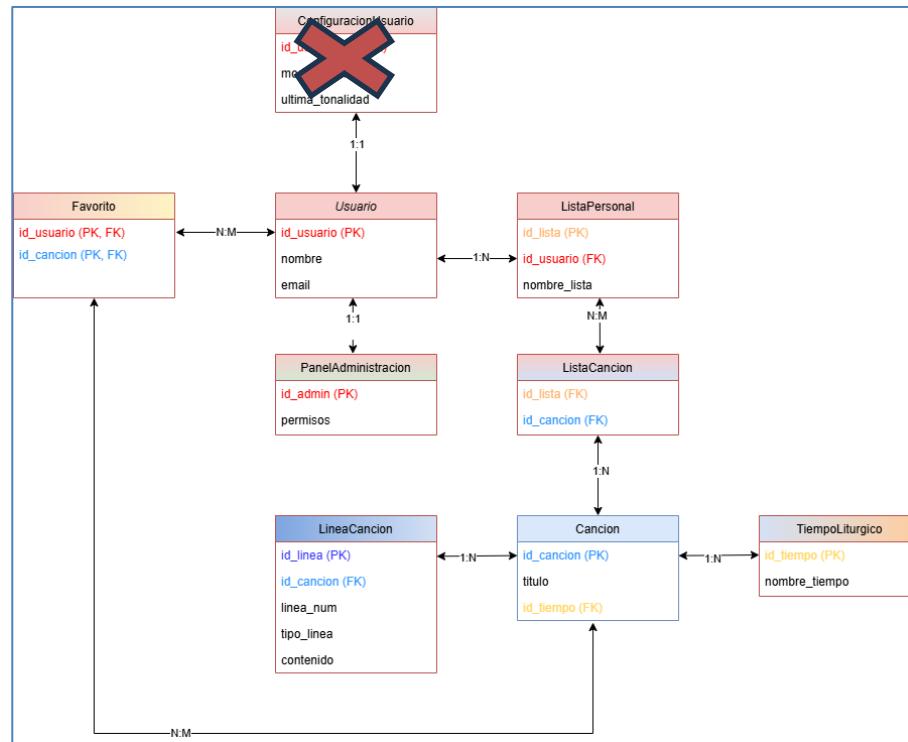
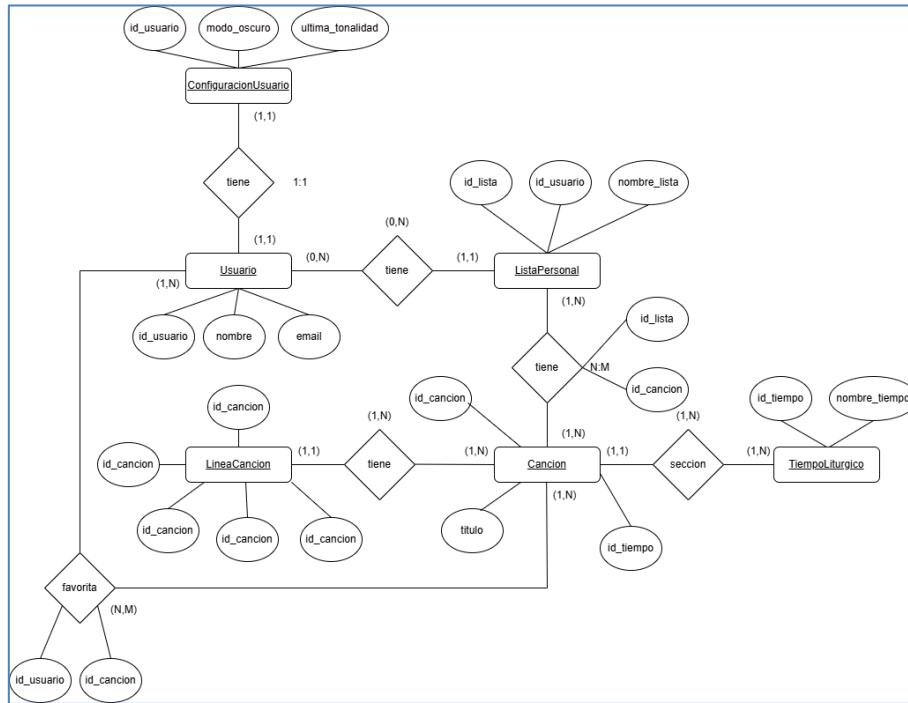
- **Sistema operativo (cliente):** Windows 10 o superior.
- **Herramientas necesarias:** las herramientas ya mencionadas en el apartado correspondiente (por ejemplo, navegador web actualizado, software de desarrollo, entorno de bases de datos, etc.).
- **Sistema operativo (servidor):** puede basarse en sistemas Linux o Windows Server, dependiendo del entorno de despliegue.

g) Restricciones

- El sistema no es compatible con versiones antiguas de navegadores, como Internet Explorer.
- Se recomienda el uso de navegadores modernos y actualizados, como Google Chrome, Mozilla Firefox o Microsoft Edge.

3. Diseño del sistema

h) Diagramas ER de bases de datos



Nota técnica: Debido a la integración de **Django Allauth** para la autenticación, el modelo de Usuario se basa en la clase extendida `AbstractUser` de Django. Esto agrega campos adicionales como `email`, `is_active`, `last_login`, entre otros. Sin embargo, las **relaciones entre entidades del sistema siguen siendo las mismas**, como se describe a continuación.

El diseño está representado en dos diagramas:

- **Diagrama Entidad-Relación (DER):** Representa las entidades y relaciones del sistema de forma conceptual.
- **Diagrama Relacional (Modelo Físico):** Muestra la estructura final de la base de datos con llaves primarias (PK) y foráneas (FK).

Entidades y Relaciones Principales

1. Usuario

- Representa a cada usuario registrado en el sistema.
- Se basa en el modelo de usuario extendido de Django (`AbstractUser`), gestionado con Allauth.
- Tiene una relación 1:1 con **ConfiguracionUsuario**.
- Puede tener 0 o muchas (**0,N**) listas personales (**ListaPersonal**).
- Puede marcar canciones como favoritas en una relación N:M con **Cancion** mediante la entidad intermedia **Favorito**.

2. ConfiguracionUsuario

- *Este modelo ha sido eliminado. Inicialmente estaba pensado para almacenar configuraciones personalizadas del usuario, como el cambio de tonalidad o el modo noche. Sin embargo, estas funcionalidades no se implementaron en la versión final del proyecto. La persistencia del cambio de tonalidad requería una estructura de datos adicional más compleja, y el modo noche no se desarrolló. Ambos casos se detallan en sus respectivos apartados de esta documentación.*
- Contiene ajustes personalizados del usuario, como **modo_oscuro** y **ultima_tonalidad**.
- Relación 1:1 con **Usuario**.

3. ListaPersonal

- Permite a los usuarios organizar sus canciones en listas personalizadas.
- Relación 1:N con **Usuario**.
- Relación N:M con **Cancion** a través de **ListaCancion**.

4. Cancion

- Contiene información de cada canción registrada.
- Tiene una relación 1:N con **TiempoLiturgico**.
- Se relaciona con **ListaPersonal** a través de **ListaCancion** (N:M).

5. TiempoLiturgico

- Representa el tiempo litúrgico al que pertenece una canción (Navidad, Pascua, etc.).
- Relación **1:N** con **Cancion**.

6. ListaCancion

- Entidad intermedia entre **ListaPersonal** y **Cancion** (N:M).

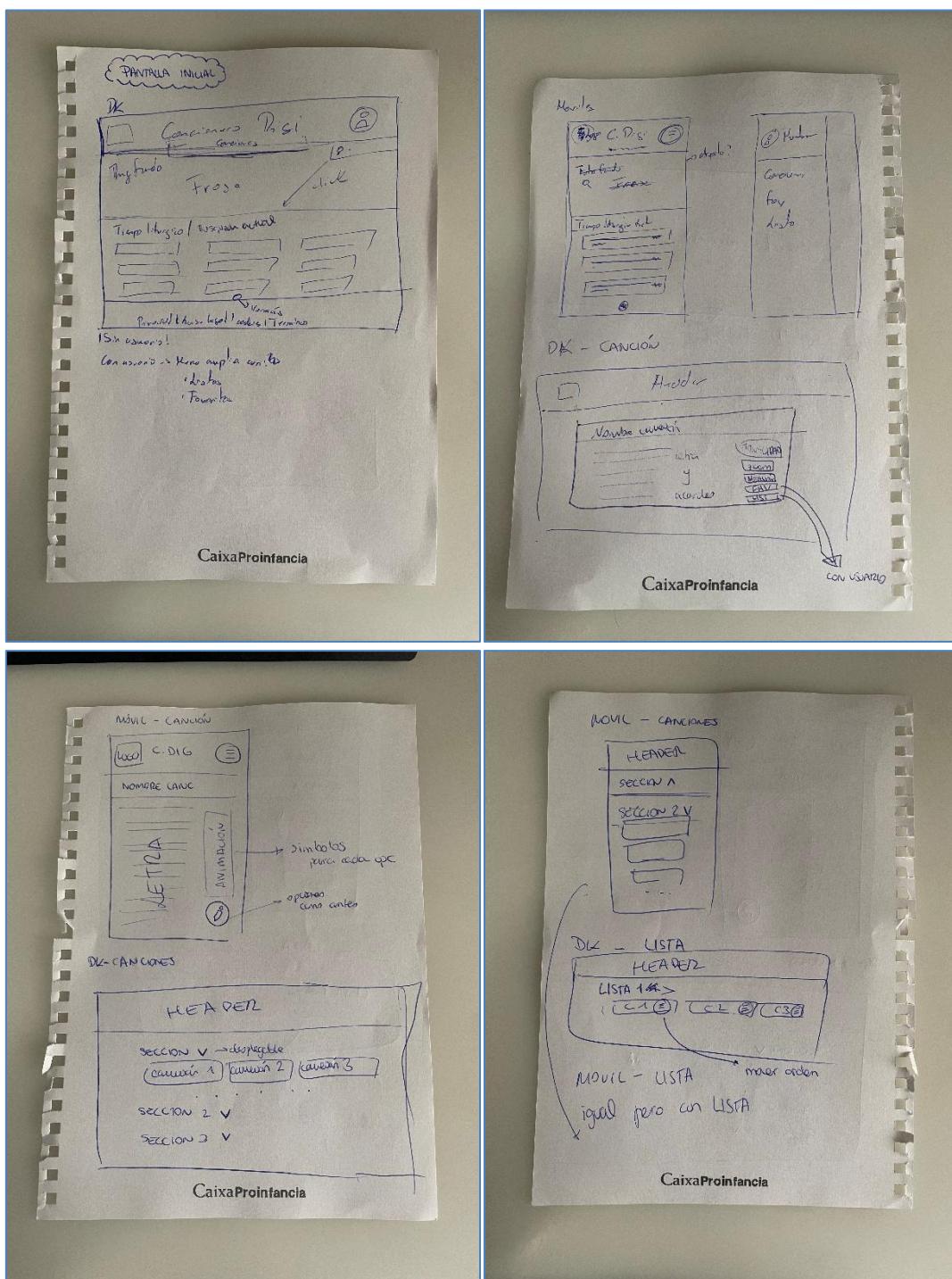
7. Favorito

- Entidad intermedia que relaciona a los **Usuarios** con sus **Canciones** favoritas en una relación N:M.

8. LineaCancion

- Representa las líneas de cada canción.
- Relación **1:N** con **Cancion**.

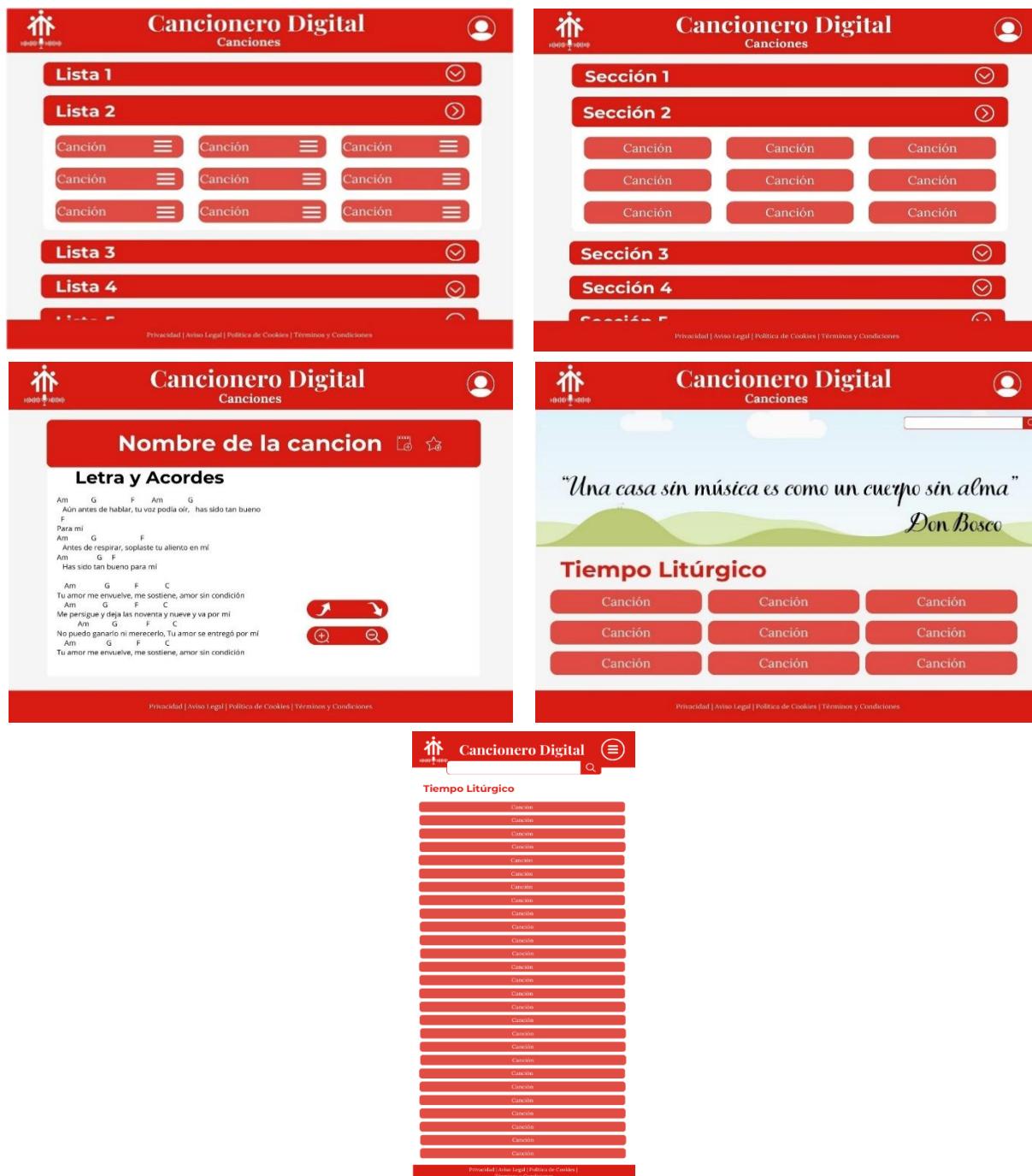
i) Diseño de la interfaz de usuario



Para el **diseño de la interfaz de usuario**, hemos realizado un boceto previo a mano para definir la estructura visual de la aplicación. En este diseño, establecimos la disposición de los elementos clave, como la navegación, las listas de canciones, las opciones de configuración del usuario y la gestión de favoritos. Estos bocetos nos ayudaron a visualizar la experiencia del usuario antes de implementar la interfaz en código.

Para el diseño de la interfaz de usuario, hemos creado un diseño más detallado utilizando la herramienta Canva, lo que nos permitió visualizar de manera más realista cómo quedaría la aplicación en diferentes dispositivos. A partir del boceto inicial, contamos con un diseño completo para escritorio y una base optimizada para la versión móvil. Estos nuevos diseños nos proporcionaron una representación más precisa de la estructura visual y la experiencia de usuario, facilitando la implementación en código.

Nota: Se decidió modificar el diseño original de las listas, ya que no resultaba visualmente adecuado ni alineado con los objetivos estéticos del proyecto. En su lugar, se implementó un nuevo formato basado en tarjetas, que ofrece una presentación más clara, moderna y coherente con el resto de la interfaz.



The image displays four wireframe prototypes of the digital hymnal application interface, arranged in a 2x2 grid. Each prototype is a red-themed interface with a header containing the logo, the title 'Cancionero Digital', and a user icon. The prototypes show different sections and designs:

- Top Left Prototype:** Shows a list view with sections labeled 'Lista 1' through 'Lista 4'. Each list item has a circular icon with a checkmark or a question mark.
- Top Right Prototype:** Shows a grid-based list view with sections labeled 'Sección 1' through 'Sección 5'. Each item is a red button-like card with the word 'Canción' repeated three times.
- Bottom Left Prototype:** Shows a search interface with a large input field labeled 'Nombre de la canción'. Below it is a section titled 'Letra y Acordes' (Lyrics and Chords) displaying musical notation and lyrics for a song.
- Bottom Right Prototype:** Shows a section titled 'Tiempo Litúrgico' (Liturgy Time) featuring a quote by Don Bosco: "Una casa sin música es como un cuerpo sin alma". It includes a grid of cards labeled 'Canción'.

j) Implementaciones.

Gestión del código con GitHub

Para el control de versiones y el trabajo colaborativo, utilizamos **GitHub** con una estrategia basada en ramas. Cada desarrollador trabajó en una rama independiente, evitando modificaciones directas en *main*.

El flujo de trabajo incluyó:

1. **Uso de GitHub Desktop** para facilitar la gestión del repositorio.
2. **Trabajo en ramas separadas** para cada funcionalidad.
3. **Sincronización periódica** con *main* mediante *pull* y *merge*.
4. **Revisión de código y Pull Requests** antes de fusionar cambios.
5. **Resolución de conflictos** cuando varias ramas afectaban el mismo archivo.

En el apartado [ANEXOS](#) se encuentra el manual que hemos empleado. Al inicio del proyecto realizamos ciertas pruebas para asentar dichos conocimientos y no surjan problemas en el futuro con código más importante.

Importación de JSON a PostgreSQL

Para poblar la base de datos en PostgreSQL, hemos desarrollado un script en Django que importa canciones desde un archivo JSON, manteniendo las relaciones entre tablas.

Estructura del JSON

```
[  
  {  
    "title": "Ejemplo de Canción",  
    "section": "Adviento",  
    "body": [  
      {"line": 1, "type": "verso", "content": "Esta es la primera línea"},  
      {"line": 2, "type": "coro", "content": "Esta es la segunda línea"}  
    ]  
  }  
]
```

Cada objeto representa una canción con su título, tiempo litúrgico y líneas.

Script de Importación

Ubicado en `canciones/management/commands/importar_canciones.py`:

Ejecución

Para importar los datos, ejecuta: `python manage.py importar_canciones`

Unificación de Tiempos Litúrgicos Duplicados en la BBDD

Objetivo:

Eliminar registros duplicados de tiempos litúrgicos en la base de datos para mejorar la consistencia y evitar confusiones en la visualización de canciones.

Motivación:

Durante la carga de datos, algunos tiempos litúrgicos se habían duplicado por diferencias en mayúsculas, acentos o caracteres especiales, lo que ocasionaba que canciones similares se asignaran a entidades distintas.

Proceso resumido:

1. Normalización de nombres:

Se limpiaron los textos para unificar criterios y detectar duplicados (eliminando acentos, ajustando mayúsculas, etc.).

2. Agrupación de duplicados:

Se agruparon los tiempos con nombres equivalentes y se eligió uno como referencia principal.

3. Reasignación de canciones:

Todas las canciones asociadas a tiempos duplicados fueron redirigidas al tiempo principal.

4. Eliminación de duplicados:

Una vez reasignadas, las entradas duplicadas fueron eliminadas de la base de datos.

```
from canciones.models import TiempoLiturgico, Cancion
import unicodedata

# Función para limpiar y normalizar texto
def limpiar_texto(texto):
    texto_normalizado = unicodedata.normalize('NFKD', texto)
    texto_normalizado = texto_normalizado.encode('ASCII', 'ignore').decode('ASCII')
    return texto_normalizado.strip().title()

# Crear un diccionario de tiempos litúrgicos normalizados
normalizados = {}

for tiempo in TiempoLiturgico.objects.all():
    nombre_normalizado = limpiar_texto(tiempo.nombre_tiempo)

    if nombre_normalizado not in normalizados:
        normalizados[nombre_normalizado] = []
    normalizados[nombre_normalizado].append(tiempo)

# Unificar los tiempos litúrgicos duplicados
for nombre_normalizado, tiempos in normalizados.items():
    if len(tiempos) > 1:
        tiempo_principal = tiempos[0]
        print(f"Unificando duplicados de: {nombre_normalizado}")

        for duplicado in tiempos[1:]:
            # Reasignar canciones

Cancion.objects.filter(id_tiempo=duplicado.id_tiempo).update(id_tiempo=tiempo_principal.id_tiempo)
            # Eliminar duplicado
            print(f"  Eliminando duplicado: {duplicado.id_tiempo} ({duplicado.nombre_tiempo})")
            duplicado.delete()

print("✅ Tiempos litúrgicos unificados correctamente.")
```

Calculo tiempo litúrgico actual

Para mostrar en la aplicación el tiempo litúrgico correspondiente a la fecha actual, se desarrollaron dos funciones en Python:

- **calcular_fecha_pascua(anio)**

Calcula la fecha del Domingo de Pascua para un año determinado utilizando el algoritmo de computus (cálculo tradicional). Esta fecha es fundamental porque determina el resto del calendario litúrgico móvil (Cuaresma, Pascua, Pentecostés, etc.).

- **obtener_tiempo_liturgico_actual()**

Determina el tiempo litúrgico actual (Adviento, Navidad, Cuaresma, Pascua, Tiempo Ordinario o Ampliación).

Lo hace comparando la fecha actual con fechas clave del calendario litúrgico (calculadas dinámicamente) como:

- Inicio de Cuaresma (46 días antes de Pascua),
- Pentecostés (49 días después de Pascua),
- Adviento (cuatro domingos antes de Navidad), etc.

Estas funciones son independientes del modelo y pueden invocarse desde las vistas para mostrar dinámicamente el tiempo litúrgico actual en la interfaz.

Funcionalidad Buscador Dinámico

Objetivo:

Implementar un sistema de búsqueda en tiempo real que permita filtrar canciones sin recargar la página, mejorando la experiencia de usuario.

Componentes principales:

- **HTML:**

Incluye un campo de entrada (`input#search-input`) y un contenedor para los resultados (`div#song-list`). El formulario (`form#search-form`) se configura para no enviarse de forma tradicional.

- **JavaScript**

(`main.js`):

Escucha el evento `input` del campo de búsqueda. Si hay 3 o más caracteres (o está vacío), envía una petición `fetch` al endpoint `/search/` usando AJAX. La respuesta contiene el HTML renderizado del listado filtrado de canciones, el cual reemplaza el contenido actual sin recarga. También actualiza el título de la sección.

- **Vista Django (search):**

- Extrae el parámetro `q` de la URL.
- Si es una petición AJAX (`XMLHttpRequest`), filtra canciones por coincidencia en el título y devuelve un fragmento HTML renderizado (`render_to_string`) dentro de un `JsonResponse`.

En caso contrario, renderiza la plantilla completa.

Funcionalidad Zoom-in y Zoom-out

Objetivo:

Permitir al usuario ajustar el tamaño del texto de las canciones mediante botones para mejorar la accesibilidad visual.

Componentes principales:

- **HTML:**

Dos botones con IDs btn-zoom-in y btn-zoom-out que afectan al tamaño de fuente de los elementos con clase .contenido-cancion.

- **JavaScript:**

Se escuchan eventos click en ambos botones. Se calcula el tamaño actual de fuente y se modifica dentro de un rango definido (minFontSize y maxFontSize). Si se alcanza alguno de los límites, el botón correspondiente se desactiva hasta que el tamaño vuelva a estar dentro del rango permitido.

Funcionalidad Solo Letra

Objetivo:

Ofrecer una vista simplificada de la canción ocultando los acordes, útil para usuarios que solo necesitan seguir la letra.

Componentes principales:

- **HTML:**

Un botón con ID btn-toggle-acordes que controla la visibilidad de los elementos con clase .acordes.

- **JavaScript:**

Al hacer clic, se alterna la visibilidad de los acordes modificando su propiedad display. Se usa una variable booleana (acordesVisibles) para determinar el estado actual. El botón actualiza su aspecto y texto según el estado.

Funcionalidad Auto-Scroll

Objetivo:

Permitir un desplazamiento automático del texto de la canción, ajustable en velocidad, para facilitar la lectura continua sin interacción manual.

Componentes principales:

- **HTML:**

- Botón btn-scroll para iniciar/detener el desplazamiento.
- Botones btn-pls-speed y btn-mn-speed para ajustar la velocidad.
- Elemento #velocidad-actual para mostrar la velocidad.
- El contenedor de la letra se referencia en JS como letraContenedor.

- **JavaScript:**

El botón btn-scroll alterna el estado del desplazamiento usando setInterval y la función scrollBy() con desplazamiento suave (behavior: "smooth").

El scroll se detiene automáticamente al llegar al final del contenido.

Los botones de velocidad modifican una variable velocidad dentro de un rango limitado, y el valor actual se muestra en tiempo real.

Uso de entorno virtual y fichero requirements.txt

Para aislar las dependencias del proyecto y facilitar su instalación en otros entornos, decidimos usar un **entorno virtual** junto con un fichero **requirements.txt**.

Pasos realizados:

1. **Creación del entorno virtual:**

```
python -m venv venv
```

2. **Activación:**

Windows: env\Scripts\activate

3. **Instalación de dependencias:**

```
pip install django psycopg2-binary django-allauth ...
```

4. **Guardar dependencias:**

```
pip freeze > requirements.txt
```

5. **Reinstalación en otros entornos:**

```
pip install -r requirements.txt
```

- **Ventajas:**

- Evita conflictos entre proyectos.
- Facilita la colaboración y el despliegue.
- Permite reproducir el entorno fácilmente.

Problemas con archivos de caché y control de versiones

Durante el desarrollo, observamos que los archivos de caché generados automáticamente por Python (`__pycache__`, .pyc, etc.) causaban molestias al realizar commits, ya que eran irrelevantes para el control de versiones.

Solución aplicada:

Creamos un archivo `.gitignore` con el siguiente contenido para ignorarlos:

```
# Ignorar archivos de caché de Python
__pycache__/
*.pyc
*.pyo
*.pyd
```

Resultado:

- Se evitaron subidas innecesarias a GitHub.
- Se mejoró la limpieza y el control del repositorio.

Elección de Django Allauth y Crispy Bootstrap5

Autenticación: django-allauth

Elegimos [django-allauth](#) como solución de autenticación completa para nuestro proyecto debido a su facilidad de integración, compatibilidad con Django y soporte nativo para autenticación social (como Google).

Además, la **documentación oficial proporciona guías detalladas** sobre cómo implementar correctamente el sistema, incluyendo configuración en settings.py, personalización de vistas y rutas, así como el uso seguro de autenticación externa. Esto nos permitió seguir buenas prácticas desde el inicio y evitar problemas comunes en entornos multiusuario.

Estilo de formularios: crispy-bootstrap5

Usamos [crispy-bootstrap5](#) para aplicar estilos visuales modernos y responsivos a nuestros formularios Django con Bootstrap 5.

La documentación incluye ejemplos claros que permitieron configurar rápidamente el renderizado de formularios mediante {{ form|crispy }}.

Personalización adicional de plantillas

A pesar de utilizar la librería crispy-bootstrap5, fue necesario **editar los templates por defecto de allauth** para que coincidieran con el estilo visual de nuestro proyecto. Esto incluyó la reorganización de secciones, incorporación de clases personalizadas y ajustes estéticos para mantener una experiencia de usuario coherente en todas las vistas de autenticación.

Panel de administración personalizado en Django.

El panel de administración de Django ha sido configurado para gestionar las entidades principales del proyecto de manera sencilla y efectiva, facilitando el control de los datos durante el desarrollo y la administración. A continuación se describen las funcionalidades específicas para cada modelo:

Canciones y Líneas de Canción

- Se permite la edición directa de las líneas asociadas a una canción desde la propia página de administración de la canción, usando un formulario en línea (TabularInline).
- En la lista principal se muestran los campos **título** y el **tiempo litúrgico** asociado a cada canción.
- Se habilita la búsqueda únicamente por el título de la canción.

Tiempos Litúrgicos

- Se permite la gestión de los tiempos litúrgicos con posibilidad de búsqueda por nombre.
- En la vista de lista se muestra únicamente el nombre del tiempo litúrgico.

Favoritos

- Se permite la gestión de las relaciones entre usuarios y canciones marcadas como favoritas.
- En la lista se visualizan el usuario y la canción correspondiente.
- Se puede buscar tanto por nombre de usuario como por título de canción.

Listas Personales

- Se permite la administración de listas personales creadas por los usuarios.
- Se muestran el nombre de la lista y el usuario que la creó.
- Se pueden buscar listas por nombre o por usuario.

Asociación Canciones-Listas

- Se permite la gestión de las relaciones entre canciones y listas personales.
- Se visualizan en la lista las asociaciones entre listas y canciones.
- Se permite búsqueda por nombre de la lista y título de la canción.

Funcionalidades de favoritos y listas personalizadas

La funcionalidad de gestión de favoritos y listas personalizadas permite a los usuarios autenticados guardar sus canciones favoritas, organizar canciones en listas personalizadas y gestionar dichas listas a través de vistas protegidas y una interfaz basada en formularios tradicionales y llamadas AJAX.

Favoritos

La vista `toggle_favorito` gestiona el marcado o desmarcado de una canción como favorita. Solo acepta solicitudes POST. El identificador de la canción se obtiene desde el cuerpo de la solicitud (`cancion_id`). Se verifica si la relación entre el usuario y la canción ya existe en el modelo `Favorito`:

- Si existe, se elimina la instancia y se responde con `{"status": "eliminado"}`.
- Si no existe, se crea una nueva instancia `Favorito` y se responde con `{"status": "agregado"}`.

Este comportamiento se maneja desde el cliente con un botón identificado como `#btn-favorito`. Al hacer clic, se envía la solicitud vía `fetch`, y según la respuesta JSON, se actualiza el contenido del botón para reflejar el nuevo estado (ícono y texto).

Para la visualización de canciones favoritas, la vista `favoritos` realiza una consulta a todos los favoritos del usuario actual, utilizando `select_related("cancion")` para optimizar el acceso a los datos relacionados. Luego, extrae únicamente las instancias `Cancion` y las pasa a la plantilla `canciones/favoritos.html`.

Listas personalizadas

La vista `toggle_list` permite crear o usar una lista existente y alternar la relación de una canción con esa lista. También se espera una solicitud POST. El formulario puede incluir un `lista_id` o un nombre para una nueva lista (`nueva_lista`). Se usa `get_or_create` para crear la lista si no existe aún. Posteriormente, se comprueba si la canción ya está en la lista:

- Si está presente, se elimina la relación (`lista.canciones.remove(cancion)`).
- Si no lo está, se añade (`lista.canciones.add(cancion)`).

Desde el cliente, esta funcionalidad se integra a través de modales Bootstrap (`modalListas`, `modalQuitarLista`) y un formulario con ID `form-quitar-lista`, que es procesado mediante JavaScript. Al enviar el formulario, se intercepta el evento, se envía una solicitud con `fetch` y se actualiza la vista según el resultado recibido ("eliminado" o "agregado").

Existe una vista adicional `guardar_cancion_en_lista`, que permite asociar una canción a una lista a través de un formulario tradicional, sin uso de AJAX. Recoge los valores `lista_id` o `nueva_lista`, crea o

obtiene la lista correspondiente, y crea la relación en la tabla intermedia (ListaCancion). Finalmente, redirige a la página de detalle de la canción.

La vista lista muestra todas las listas personales creadas por el usuario autenticado. Se construye una estructura de datos que contiene cada lista y las canciones asociadas, utilizando Cancion.objects.filter(listacancion__lista=list). Esta información se pasa a la plantilla canciones/lista.html, donde se presenta en formato de tarjetas, mostrando hasta 4 canciones por lista y una insignia si hay más canciones.

La vista lista_detalle muestra una lista específica con todas sus canciones asociadas. Esta vista asegura que el usuario es propietario de la lista consultada. En la plantilla canciones/lista_detalle.html, se ofrece también un formulario de exportación a Word, con opción de incluir acordes.

Consideraciones adicionales

- Todas las vistas están protegidas con el decorador @login_required.
- Se utiliza get_object_or_404 para validar y acceder de forma segura a objetos basados en el usuario autenticado.
- Las interacciones en el frontend utilizan fetch para evitar recargas innecesarias y mejorar la experiencia de usuario.
- Los tokens CSRF y las URLs de destino se inyectan como variables JavaScript (window.toggleFavoritoUrl, window.csrfToken, etc.) desde las plantillas.
- El sistema de alertas tras acciones se gestiona mediante alert() y recarga de página cuando es necesario.

Exportación de listas personalizadas a Word

Los usuarios autenticados pueden exportar cualquier lista de canciones propia a un archivo .docx. Esta funcionalidad está disponible desde la vista de detalle de una lista personalizada y permite incluir opcionalmente los acordes de las canciones.

Es necesario instalar la librería python-docx para que esta funcionalidad esté disponible. Puedes hacerlo ejecutando: `pip install python-docx`

Descripción de la funcionalidad

La vista exportar_lista_word genera un archivo .docx que contiene el contenido de todas las canciones asociadas a una lista específica del usuario autenticado. Este documento incluye los títulos de las canciones, sus líneas en orden y un formato especial para acordes y estribillos.

El flujo completo de esta función es el siguiente:

- Se obtiene la lista correspondiente, asegurándose de que pertenezca al usuario actual mediante get_object_or_404.
- Se verifica si se deben incluir acordes, leyendo el parámetro acordes=1 desde la URL (request.GET).
- Se accede a todas las canciones de la lista a través de la relación intermedia ListaCancion, usando select_related("cancion") para eficiencia.
- Se construye el documento Word con python-docx, agregando:

- Un encabezado con el nombre de la lista.
- Un subtítulo con el título de cada canción.
- Las líneas de cada canción en orden (linea_num), omitiendo los acordes si el usuario no los ha solicitado.
- Las líneas de tipo acorde y estribillo se muestran en negrita.
- Si la lista no contiene canciones, se añade un mensaje indicativo en el documento.
- Finalmente, el documento se devuelve como archivo descargable con el nombre de la lista.

Integración en la plantilla

Desde la plantilla canciones/lista_detalle.html, se presenta un formulario en línea para exportar la lista actual. Este formulario:

- Tiene un checkbox marcado por defecto para incluir acordes (name="acordes" value="1").
- Al enviarse, redirige a la URL asociada a exportar_lista_word, pasando el parámetro correspondiente en la query string.
- La descarga del archivo generado es automática, y el nombre del archivo se forma con el nombre de la lista (<nombre_lista>.docx).

Consideraciones adicionales

- El uso de python-docx permite estructurar el contenido con encabezados, párrafos y estilos de texto enriquecido de manera sencilla.

Transposición de acordes

El sistema permite transponer acordes musicales de una canción hacia arriba o abajo en semitonos, lo cual es útil para adaptar la tonalidad a la voz del cantante o al instrumento utilizado.

Esta funcionalidad se aplica dinámicamente en la vista de detalle de canción, leyendo un parámetro transpose en la URL. Las líneas marcadas como acordes (tipo_linea = 'acorde') se modifican automáticamente según el número de semitonos indicado.

Para ello, se define una lista de notas musicales en español con sostenidos (#), y se trabaja con sus índices para aplicar la transposición. También se implementa una conversión adicional para evitar resultados poco musicales, como dobles sostenidos (por ejemplo, FA##), que se reemplazan por su equivalente enarmónico más común (SOL), utilizando el diccionario ENHARMONICAS_SIMPLIFICADAS.

IV. CONCLUSIÓN

A lo largo del desarrollo de este proyecto se han alcanzado con éxito los objetivos propuestos inicialmente, entre los cuales destacan:

- La creación de una **aplicación funcional para la gestión y visualización de un repertorio musical litúrgico**, especialmente adaptado al contexto salesiano.
- La posibilidad de **gestionar usuarios, canciones y listas personalizadas**, permitiendo a los administradores añadir, editar y organizar contenido con facilidad.
- La implementación de funciones clave como la **transposición automática de acordes**, facilitando su uso por músicos con distintas capacidades vocales o instrumentales.
- El uso de tecnologías actuales y accesibles, con una **interfaz clara y usable**, pensada para facilitar la adopción por parte de distintos tipos de usuarios (músicos, animadores, educadores...).

Durante el desarrollo, se han respetado las limitaciones iniciales en cuanto a simplicidad y facilidad de mantenimiento, optando por soluciones técnicas realistas dentro del tiempo y recursos disponibles. Se ha priorizado una arquitectura limpia, modular y fácilmente ampliable en el futuro.

Propuestas de mejora futuras

Durante el desarrollo del proyecto se han detectado varias posibles mejoras que, por razones de tiempo y priorización, no han podido implementarse pero que aportarían un valor significativo en futuras versiones:

- **Orden personalizado de canciones en las listas:**
Actualmente, las canciones dentro de una lista se muestran en el orden en que fueron añadidas. Sería deseable permitir al usuario reordenarlas manualmente, añadiendo un campo adicional en la tabla intermedia que relacione listas y canciones. Esto mejoraría la experiencia al permitir una secuencia lógica o litúrgica personalizada.
- **Subida de versiones cantadas por usuarios:**
Una funcionalidad clave para generar comunidad sería permitir que los usuarios puedan grabar y subir su propia versión cantada de las canciones del repertorio. Los administradores podrían seleccionar qué versiones publicar y destacar. De este modo, se fomenta la participación, el aprendizaje mutuo y el espíritu comunitario característico de los ambientes salesianos.
- **Modo oscuro:**
Aunque no se considera una necesidad principal (dado que las celebraciones suelen darse en entornos diurnos), el modo oscuro podría beneficiar la visualización en ciertos contextos o facilitar la lectura en dispositivos por la noche. Por falta de tiempo, no se ha llegado a implementar, pero sería fácilmente integrable a nivel de frontend con cambios de CSS o mediante frameworks que soporten temas dinámicos.
- **Compatibilidad con notación anglosajona de acordes:**
La lógica actual de transposición y detección de acordes ha sido diseñada para notación

latina (DO, RE, MI...), por lo que aplicar directamente la notación anglosajona (C, D, E...) implicaría reestructurar parte del sistema de reconocimiento y transposición, incluyendo traducción acorde por acorde en cada línea. Esto podría causar sobrecarga en el procesamiento y pérdida de rendimiento si no se implementa de forma eficiente. No obstante, es una mejora muy útil, sobre todo pensando en usuarios con formación musical internacional.

Otras mejoras posibles a contemplar

- **Funcionalidad offline (modo sin conexión):**

Para garantizar el uso en contextos con conectividad limitada (por ejemplo, durante acampadas, retiros o celebraciones en exteriores), sería útil incorporar una versión offline, al menos para la visualización del repertorio previamente descargado.

- **Sistema de búsqueda avanzada y filtrado:**

Aunque ya existe un buscador básico, se podría ampliar permitiendo filtrar por autor, tema litúrgico, tono, uso recomendado, entre otros. Esto facilitaría el acceso rápido a canciones según las necesidades del momento.

- **Internacionalización:**

Actualmente, el sistema está pensado para usuarios hispanohablantes. Adaptarlo para otros idiomas (o al menos permitir traducción de la interfaz) abriría el uso a comunidades salesianas en otros países.

- **Integración con dispositivos móviles y tablets:**

Aunque ya es accesible desde navegador, una versión responsive optimizada o una aplicación móvil nativa mejoraría significativamente la usabilidad durante celebraciones o ensayos.

- **Historial de uso y estadísticas:**

Incorporar estadísticas básicas (por ejemplo, canciones más usadas, listas más consultadas, etc.) permitiría mejorar el repertorio y tomar decisiones informadas por parte de los responsables litúrgicos.

Estado del despliegue

Dado que el proyecto nace con una clara **motivación social y comunitaria**, el objetivo principal no ha sido simplemente completar una aplicación técnica, sino iniciar un camino hacia una herramienta cómoda, accesible y útil para su entorno específico de uso. En este sentido, se considera que **el despliegue de la aplicación no es prioritario por el momento**, por dos razones principales:

1. **El proyecto seguirá en desarrollo activo**, en colaboración con un compañero del centro juvenil que ha participado en el desarrollo de una aplicación previa sobre la que se inspira parcialmente este trabajo. El objetivo es continuar implementando mejoras y nuevas funcionalidades con una visión compartida a medio plazo.
2. **Las limitaciones técnicas y económicas del despliegue**, especialmente en lo relativo al uso de PostgreSQL y Django en servidores gratuitos, suponen una barrera práctica para una publicación inmediata. No obstante, se adjunta un manual técnico que documenta detalladamente los pasos necesarios para llevar a cabo el despliegue cuando se considere oportuno, accesible en el siguiente enlace: [Manual para Desplegar una Aplicación Django con PostgreSQL](#)

V. BIBLIOGRAFÍA

Backend con Django

- [**Documentación oficial de Django**](#)
Referencia principal para el desarrollo backend. Ha sido utilizada en todos los aspectos relacionados con la estructura del proyecto: definición de modelos, vistas, plantillas, sistema de autenticación, gestión de formularios y otros elementos esenciales del framework.
- [**Django QuerySet API Reference**](#)
Documentación específica para la gestión de consultas a base de datos mediante el ORM de Django. Fue de especial utilidad para comprender y optimizar operaciones como filtrado, ordenación, actualización masiva y relaciones entre modelos.
- [**Uso de comandos personalizados en Django**](#)
Guía utilizada para implementar comandos de administración que automatizan tareas, como la actualización del calendario litúrgico o la importación masiva de canciones desde archivos estructurados.
- [**Django Allauth - Documentación oficial**](#)
Documentación empleada para configurar el sistema de autenticación con funcionalidades avanzadas como el registro, la verificación por correo electrónico y el inicio de sesión mediante cuentas sociales. También se utilizó la [sección de configuración de cuentas](#) para personalizar la experiencia del usuario.

Interacción asíncrona y desarrollo del frontend

- [**Django AJAX Integration \(Simple is Better Than Complex\)**](#)
Tutorial clave para la implementación de AJAX con Django. Facilitó el desarrollo de funcionalidades como la transposición dinámica de acordes sin recarga de página.
- [**JavaScript Fetch API \(MDN\)**](#)
Referencia para el uso de la API Fetch en JavaScript. Se empleó en la comunicación asíncrona entre el frontend y el backend para peticiones como el cambio de tonalidad o la carga parcial de contenidos.
- [**Manipulación del DOM y gestión de eventos en JavaScript \(MDN\)**](#)
Utilizada para desarrollar interacciones dinámicas en la interfaz: zoom, scroll automático, despliegue de acordes y otros elementos de experiencia de usuario.
- [**Bootstrap Icons**](#)
Catálogo consultado para integrar iconografía clara en botones de navegación, accesibilidad y funciones interactivas.

Procesamiento musical y textual

- [**Expresiones regulares en Python \(re module\)**](#)

Referencia indispensable para la creación de patrones utilizados en la transposición automática de acordes, permitiendo identificar estructuras musicales dentro de líneas de texto.

- [**Unicode y normalización de cadenas en Python \(unicodedata\)**](#)

Se consultó para evitar duplicados en nombres mediante la estandarización de acentos, caracteres especiales y mayúsculas/minúsculas.

- [**Algoritmo para el cálculo de la fecha de Pascua \(Computus\)**](#)

Base teórica para la función que calcula la fecha de Pascua, a partir de la cual se derivan otras festividades móviles en el calendario litúrgico.

Manipulación de datos estructurados

- [**Manejo de archivos JSON en Python**](#)

Se utilizó para importar y exportar datos estructurados (como canciones, listas, configuraciones), permitiendo una gestión versátil de contenidos persistentes.

Estilo y buenas prácticas

- [**Guía de buenas prácticas en comentarios de código \(Real Python\)**](#)

Recurso consultado para mantener un estilo claro y coherente en los comentarios del código, ayudando a su legibilidad, documentación y mantenimiento.

Asistencia mediante modelos de lenguaje

- [**OpenAI ChatGPT**](#)

Se utilizó como herramienta de apoyo en diversas fases del desarrollo. Su uso fue especialmente relevante para:

- Resolver errores específicos de sintaxis, lógica o configuración.
- Documentar funciones complejas de forma concisa y estructurada.
- Sugerir mejoras en la organización del código y en el diseño de funcionalidades.
- Comentar bloques de código de forma comprensible para otros desarrolladores o lectores de la memoria.

Recursos adicionales

- [**Playlist en YouTube sobre Django Allauth y autenticación de usuarios**](#)

Colección de vídeos consultada como apoyo visual para la integración de Django Allauth, autenticación social y gestión de usuarios.

VII. GLOSARIO DE TÉRMINOS

A continuación se presenta un glosario de términos técnicos y específicos utilizados a lo largo de este trabajo. El objetivo es facilitar la comprensión del proyecto incluso para lectores que no estén familiarizados con el entorno de desarrollo web o con la música litúrgica.

A

- **AJAX (Asynchronous JavaScript And XML):** Técnica de desarrollo web que permite enviar y recibir datos del servidor en segundo plano, sin necesidad de recargar completamente la página. En este proyecto se utilizó para mejorar la experiencia de usuario, por ejemplo, al cambiar tonalidades o añadir canciones a listas.
- **Acorde:** Conjunto de notas musicales que suenan simultáneamente y forman la base armónica de una canción. En el cancionero, los acordes se muestran junto a la letra y pueden transponerse dinámicamente.
- **Administración (Panel de):** Interfaz destinada a usuarios con permisos especiales, desde donde se pueden gestionar contenidos como canciones, listas, tiempos litúrgicos y usuarios.
- **Allauth (Django Allauth):** Biblioteca de Django para implementar autenticación de usuarios con funciones como login con Google, verificación por correo electrónico y gestión de cuentas.
- **API (Application Programming Interface):** Conjunto de reglas que permiten la comunicación entre diferentes programas. En este proyecto se utiliza indirectamente a través de la API fetch de JavaScript.
- **Autenticación:** Proceso mediante el cual un usuario se identifica en el sistema. En este proyecto se realiza a través de cuentas de Google (OAuth).

B

- **Backend:** Parte del software que gestiona la lógica de negocio, acceso a base de datos y funcionamiento del servidor. En este proyecto, se implementó con Django (Python).
 - **Bootstrap:** Framework de diseño CSS utilizado para estilizar interfaces web de forma rápida y responsive. Se usó junto a crispy-bootstrap5 para mejorar la estética de formularios y otros elementos de la interfaz.
-

C

- **Canción (entidad):** Elemento principal de la aplicación, compuesto por una o varias líneas (verso, estribillo, acorde), título y clasificación litúrgica.
 - **CSS (Cascading Style Sheets):** Lenguaje usado para describir la presentación de documentos HTML, permitiendo definir colores, tamaños, márgenes, y estilos visuales.
 - **CSV (Comma-Separated Values):** Aunque no se usó en este proyecto, es un formato común para importar/exportar datos en texto plano, similar al JSON usado.
 - **Crispy-bootstrap5:** Extensión para Django que permite renderizar formularios con estilo de Bootstrap 5 de manera sencilla y coherente.
-

D

- **Django:** Framework de desarrollo web en Python que facilita la creación de aplicaciones robustas mediante el uso de modelos, vistas, plantillas y un ORM integrado.
 - **DOM (Document Object Model):** Representación estructurada de un documento HTML. JavaScript interactúa con el DOM para modificar dinámicamente el contenido de la página.
 - **Docx (formato):** Extensión de documentos de Microsoft Word. En este proyecto, las listas personalizadas pueden exportarse en este formato mediante la librería python-docx.
-

E

- **Entorno Virtual (venv):** Herramienta de Python que permite aislar dependencias del proyecto, evitando conflictos entre paquetes de distintos proyectos.
 - **Espacio litúrgico / Tiempo litúrgico:** Clasificación temporal de celebraciones religiosas (Adviento, Cuaresma, Pascua, etc.). En el cancionero se usa para categorizar las canciones.
-

F

- **Fetch API:** Interfaz moderna de JavaScript utilizada para hacer peticiones HTTP asíncronas. Sustituye a métodos más antiguos como XMLHttpRequest.
 - **Frontend:** Parte visual de una aplicación con la que interactúa el usuario. En este proyecto incluye HTML, CSS y JavaScript.
 - **Framework:** Conjunto de herramientas y reglas que facilitan el desarrollo de software. Django y Bootstrap son ejemplos de frameworks utilizados.
 - **Formulario:** Elemento HTML que permite recoger datos del usuario. Se emplea para autenticar usuarios, crear listas y más.
-

G

- **Git:** Sistema de control de versiones que permite gestionar los cambios realizados en el código a lo largo del tiempo.
 - **GitHub:** Plataforma basada en Git que permite colaboración en proyectos de desarrollo mediante repositorios, ramas y pull requests.
 - **Google OAuth:** Sistema de autenticación que permite iniciar sesión con una cuenta de Google sin necesidad de gestionar contraseñas manualmente.
-

H

- **HTML (HyperText Markup Language):** Lenguaje principal para estructurar el contenido de páginas web. Utilizado como base del frontend.
-
-

I

- **Interfaz de usuario (UI):** Medio a través del cual un usuario interactúa con una aplicación. En este proyecto, se diseñó una interfaz amigable para facilitar la consulta y organización de canciones.
-

J

- **JavaScript:** Lenguaje de programación utilizado para hacer páginas web interactivas. En este proyecto se emplea para funciones como scroll automático, zoom, filtros y peticiones asíncronas.
 - **JSON (JavaScript Object Notation):** Formato ligero de intercambio de datos. Usado para importar canciones a la base de datos y facilitar exportaciones.
-

L

- **Lista personalizada:** Conjunto de canciones agrupadas por un usuario para un fin específico. Las listas pueden ser nombradas, modificadas, exportadas o eliminadas.
 - **Login:** Proceso de acceso de un usuario a la plataforma mediante credenciales o cuenta externa (en este caso, Google).
-

M

- **Markdown:** Lenguaje de marcado ligero que permite escribir texto con formato (negritas, listas, títulos) de forma simple. Se menciona como opción futura para exportar listas.
 - **Modo oscuro:** Configuración visual opcional en aplicaciones que invierte los colores para reducir el esfuerzo visual en ambientes oscuros. Fue considerado como mejora futura.
-

N

- **Notación musical (latina/anglosajona):** Sistemas para nombrar las notas musicales. El sistema latino (DO, RE, MI...) se usó en el proyecto; el anglosajón (C, D, E...) se propuso como mejora futura.
 - **Normalización de texto:** Proceso de eliminación de acentos y caracteres especiales para evitar duplicados en la base de datos y mejorar la consistencia.
-

O

- **ORM (Object Relational Mapper):** Herramienta que permite trabajar con bases de datos usando objetos en lugar de escribir consultas SQL manuales. Django incluye un ORM propio.
-
-

P

- **Panel de administración:** Interfaz interna de Django para gestionar los modelos del sistema como usuarios, canciones o tiempos litúrgicos.
 - **PostgreSQL:** Sistema de gestión de bases de datos relacional y de código abierto utilizado para almacenar las canciones, usuarios, listas, etc.
 - **Python:** Lenguaje de programación principal del backend del proyecto, especialmente usado con Django.
 - **Python-docx:** Biblioteca de Python para crear y manipular documentos de Word, usada para exportar listas en formato .docx.
-

R

- **Renderizado:** Proceso por el cual se genera una vista visual de los datos en el navegador. Puede ser completo o parcial (en tiempo real vía AJAX).
 - **Repositorio:** Lugar donde se almacena y gestiona el código fuente de un proyecto, generalmente en plataformas como GitHub.
-

S

- **Scroll automático:** Función que permite desplazar el contenido verticalmente de forma continua, útil para seguir la letra de una canción durante una interpretación sin usar las manos.
 - **Sistema litúrgico:** Estructura del calendario religioso católico que divide el año en tiempos litúrgicos y fiestas. Este sistema sirve de base para clasificar las canciones.
-

T

- **Tonalidad:** Conjunto de características musicales que definen la altura base de una canción. En la aplicación se puede modificar con la función de transposición.
 - **Transposición:** Cambio de la tonalidad de una canción, modificando todos sus acordes para ajustarse a una nueva altura. Funcionalidad incluida en tiempo real.
 - **Trello:** Herramienta de gestión de tareas basada en tableros, utilizada para organizar el trabajo del proyecto durante su desarrollo.
-

U

- **UI/UX (User Interface / User Experience):** Diseño de interfaz y experiencia de usuario. En este proyecto se diseñó pensando en la accesibilidad y facilidad de uso en celebraciones religiosas.
-

- **Unicode:** Estándar para representar caracteres de cualquier idioma. Se usó para eliminar inconsistencias en nombres con tildes o símbolos especiales.
 - **Usuario autenticado:** Usuario que ha iniciado sesión en el sistema. Tiene acceso a funcionalidades como listas personalizadas y favoritos.
-

V

- **Venv (Virtual Environment):** Entorno virtual de Python que permite gestionar dependencias sin afectar otros proyectos.
 - **Vista (Django):** Función o clase que responde a una solicitud web y devuelve una respuesta. Las vistas gestionan la lógica de cada sección de la app.
-

Z

- **Zoom:** Funcionalidad que permite aumentar o reducir el tamaño de letra de las canciones para mejorar la legibilidad en distintas situaciones.

● ANEXOS

[1. Manual Control de Versiones en GitHub.](#)