UNIVERSIDAD
DE GRANADA

Diseño y Desarrollo de Sistemas de Información

Diseño de un sistema de información para la gestión y reproducción de música.

Entrega final.

Darío Abad Tarifa
Juan Francisco Díaz Moreno
Pedro Domínguez López
Javier Sáez de la Coba

Curso 2018-2019

20 de enero de 2019

Índice

1. Descripción del problema a resolver. 2

2. Análisis de requisitos. 2
- 2.1. Requisitos de datos . . . 2
- 2.2. Requisitos funcionales . . . 8
- 2.3. Validación cruzada de requisitos . . . 11

3. Diseño. 13
- 3.1. Esquema de caja negra. . . . 13
- 3.2. Esquema armazón DFD0. . . . 13
- 3.3. Esquemas externos del DFD0. . . . 14
  - 3.3.1. EE0 Módulo social . . . 14
- 3.4. Refinamientos. . . . 14
  - 3.4.1. Módulo escuchar canciones. . . . 14
  - 3.4.2. Módulo gestionar canciones. . . . 17
  - 3.4.3. Módulo gestión de usuarios. . . . 20
  - 3.4.4. Módulo social. . . . 21
  - 3.4.5. Unión de módulos DFD1. . . . 25
  - 3.4.6. Unión de módulos DFD2. . . . 25
- 3.5. Esquema entidad relación. . . . 26
- 3.6. Paso a tablas. . . . 26

4. Implementación. 28
- 4.1. Diseño físico de la base de datos. . . . 28
  - 4.1.1. Sentencias SQL. . . . 29
- 4.2. Implementación modulo escuchar música. . . . 33
  - 4.2.1. Implementación del disparador. . . . 33
  - 4.2.2. Implementación de añadir canción a una lista. . . . 33
- 4.3. Implementación modulo social. . . . 38
  - 4.3.1. Implementación del disparador. . . . 38
  - 4.3.2. Implementación de un proceso. . . . 38
- 4.4. Valoración de canciones. . . . 42
  - 4.4.1. Implementación del disparador. . . . 42
  - 4.4.2. Implementación de un proceso. . . . 43

5. Conclusiones 51

1. Descripción del problema a resolver.

Una empresa de streaming de audio quiere rehacer su plataforma de gestión de música (todo el servicio excepto el propio streaming de música). Para ello requiere que puedan haber cuentas de oyentes y de artistas.

El sistema ofrece un servicio de música en streaming, es decir permite que un usuario pueda buscar una canción, un álbum o una lista de música en la base de datos para posteriormente escucharla.

Las listas funcionan como si de un directorio se tratase, el usuario las gestiona como quiere y en ellas guarda sus canciones favoritas o canciones según un tema especifico.

El sistema ofrece ademas diferentes opciones de carácter social para facilitar la relación entre amigos y los artistas con sus oyentes. Un usuario puede seguir a otro usuario (sea artista o no) para estar al corriente de lo que escucha proporcionando el nombre del usuario al que quiere seguir. Un usuario podrá recomendar una canción a otro usuario proporcionando el nombre de este y el identificador de la canción que quiere recomendar. Un usuario podrá solicitar un resumen sobre las canciones que escucha su lista de amigos. Un usuario podrá ver las recomendaciones que le llegan de sus amigos. Además, los usuarios oyentes pueden valorar las canciones que escuchan. Por último podrán visualizar las canciones mejor valoradas por un usuario proporcionando su nombre de usuario.

2. Análisis de requisitos.

2.1. Requisitos de datos

RD1 Identificador de la canción, álbum o lista para escuchar:

- Elección (canción, álbum o lista)
- Identificador

RD2 Datos de álbum almacenado:

- Nombre del álbum
- Nombre del artista
- Fecha de introducción
- Identificador del álbum
- Nombre del álbum
- Nombre del artista

- Fecha de introducción
- Número de canciones
- Duración

RD3 Datos de canción almacenada:

- Identificador de la canción
- Nombre de la canción
- Nombre del artista
- Nombre del álbum
- Estilo de la canción
- Duración de la canción
- Fecha de introducción
- Archivo de audio
- Número de reproducciones
- Valoración media

RD4 Datos de lista almacenada:

- Identificador de la lista
- Nombre de la lista
- Canciones que contiene
- Duración de la lista
- Fecha de creación
- Usuario al que pertenece
- Seguidores

RD5 Identificador de la nueva lista creada:

- Identificador de la lista

RD6 Nombre para la búsqueda de una canción, álbum o lista:

- Nombre

RD7 Lista con los posibles identificadores según el resultado de la búsqueda:

- Identificador
- Tipo: canción, álbum, lista
- Nombre

- Usuario o artista
- Fecha de publicación o de creación
- Duración

RD8 Datos para crear una lista nueva:

- Usuario
- Fecha

RD9 Valoración de una canción:

- Identificador de la canción
- Identificador

RD10 Datos nuevo álbum:

- Nombre del álbum
- Nombre del artista
- Fecha de introducción

RD11 Datos nueva canción:

- Nombre de la canción
- Nombre del artista
- Nombre del álbum
- Estilo de la canción
- Duración de la canción
- Fecha de introducción
- Archivo de audio

RD12 Identificador de la canción introducida:

- Identificador de la canción

RD13 Identificador de la canción para ver sus estadísticas:

- Identificador de la canción

RD14 Reproduccion de una lista:

- Audio de las canciones de la lista

RD15 Estadísticas de la canción:

- Nombre de la canción

- Número de reproducciones
- Valoración media

RD16 Identificador de la canción para añadirla a canciones destacadas:

- Identificador de la canción

RD17 Lista de las canciones destacadas actualizada:

- Canciones destacadas

RD18 Identificador de la canción o álbum para eliminarlo:

- Elección (canción o álbum)
- Identificador

RD19 Datos de registro del usuario:

- Nombre de usuario
- Correo electrónico
- Contraseña
- Nombre
- Apellidos
- Tipo de usuario (oyente, artista)
- Dirección

RD20 Datos usuario almacenado:

- Identificador
- Nombre de usuario
- Correo electrónico
- Contraseña
- Nombre
- Apellidos
- Tipo de usuario (oyente, artista)
- Dirección

RD21 Contraseña nueva:

- Identificador de usuario
- Contraseña nueva

RD22 Datos perfil usuario:

- Nombre de usuario
- Correo electrónico
- Contraseña
- Nombre
- Apellidos
- Dirección

RD23 Identificador del usuario a eliminar:

- Identificador del usuario

RD24 Datos de un usuario:

- Nombre de usuario

RD25 Datos de un amigo:

- Nombre de usuario
- Canción que está escuchando

RD26 Datos de artista

- Nombre de usuario
- Nombre artístico
- Número de canciones subidas
- Álbumes subidos

RD27 Recomendación saliente

- Nombre de usuario de salida
- Cuerpo de mensaje
- Nombre de usuario de entrada

RD28 Recomendación entrante

- Nombre de usuario de entrada
- Cuerpo de mensaje
- Nombre de usuario de salida

RD29 Recomendación almacenada.

- Nombre de usuario que recomienda
- Cuerpo de mensaje

- Nombre de usuario que es recomendado

RD30 Datos de canción que está siendo escuchada

- Identificador de canción
- Usuario que la está escuchando

RD31 Datos del usuario actual

- Identificador de usuario

RD32 Datos de canciones mejor valoradas

- Identificadores de canciones
- Identificador de usuario que ha valorado

RD33 Datos de usuario que ha realizado valoraciones

- Nombre de usuario

RD34 Valoración dada por un usuario a una canción

- identificador de canción
- Valoración

RD35 Datos modificar estado de canción en lista.

- Identificador de canción
- Identificador de la lista de reproducción
- Operación a realizar

RD36 Identificador de la lista a eliminar

- Identificador de la lista de reproducción

RD37 Datos canciones que están siendo escuchadas

- Identificador de la canción
- Identificador de usuario que escucha la canción

RD38 Datos recomendaciones almacenadas

- Identificador del usuario que recomienda
- Identificador del usuario que recibe la recomendación
- Identificador de la canción que es recomendada

RD39 Datos canciones mejor valoradas

- Nombre de la canción
- Nombre del artista
- Valoración

RD40 Identificador de un álbum para eliminar

- Identificador del álbum a eliminar

RD41 Identificador de álbum introducido

- Identificador del álbum a eliminar

2.2. Requisitos funcionales

RF1 Escuchar una canción: un usuario introduce el identificador de la canción, álbum o lista que quiere escuchar..

- Entrada: RD1
- Consulta: RD2, RD3, RD4
- Salida: RD14

RF2 Buscar una canción: a través del nombre de una canción, álbum o lista se elige el identificador que se considere el adecuado con los criterios de búsqueda .

- Entrada: RD6
- Consulta: RD2, RD3, RD4
- Salida: RD7

RF3 Crear listas de reproducción: un usuario puede crear una lista donde añade canciones y las tiene todas en un mismo lugar. .

- Entrada: RD8
- Almacena: RD4
- Salida: RD5

RF4 Añadir canciones a una lista de reproducción: un usuario puede añadir o eliminar las canciones de su lista a traves del identificador..

- Entrada: RD35
- Consulta: RD4
- Actualiza: RD4

RF5 Borrar lista: un usuario puede eliminar su lista..

- Entrada: RD36
- Consulta: RD4
- Actualiza: RD4

RF6 Valorar canciones: un usuario puede valorar una canción..

- Entrada: RD34
- Consulta: RD9
- Actualiza: RD9

RF7 Crear álbum: un artista registra en el sistema un nuevo álbum..

- Entrada: RD10
- Almacena: RD2
- Salida: ninguna

RF8 Crear canción: un artista introduce en el sistema una nueva canción..

- Entrada: RD11
- Consulta: RD2
- Actualiza: RD3
- Salida: RD12

RF9 Ver estadísticas de canción: un artista consulta las estadísticas de una de sus canciones buscándola con su identificador..

- Entrada: RD13
- Consulta: RD12
- Salida: RD15

RF10 Destacar canciones individuales: un artista selecciona una canción a través de su identificador y la añade a su lista de canciones destacadas..

- Entrada: RD16
- Consulta: RD3
- Salida: RD17

RF11 Borrar canciones: un artista introduce en el sistema si quiere eliminar una canción o un álbum y el identificador de su elección y el sistema elimina los datos asociados al mismo..

- Entrada: RD18
- Consulta: RD2 RD3

- Actualiza: RD2 RD3

RF12 Crear usuarios. El sistema guarda la información de un nuevo usuario .

- Entrada: RD19
- Almacena: RD20

RF13 Recuperar contraseña: .

- Entrada: RD21
- Actualiza: RD20

RF14 Modificar perfil. El sistema guarda la información modificada por el usuario. .

- Entrada: RD22
- Almacena: RD20

RF15 Borrar usuarios. Eliminar a un usuario del sistema..

- Entrada: RD23
- Actualiza: RD20

RF16 Seguir usuario. Se crea una relación entre dos usuarios, el seguidor estará al tanto de las acciones del usuario al que sigue.

- Entrada: RD24
- Almacenado: RD26

RF17 Recomendar canciones a amigos. Un usuario envía a través de un mensaje una canción a otro usuario.

- Entrada: RD27
- Almacenado: RD29

RF18 Ver lo que están escuchando los amigos. Se muestra al usuario una lista con las canciones que están escuchando sus amigos en ese momento.

- Entrada: RD25
- Consulta: RD37
- Salida: RD30

RF19 Ver recomendaciones entrantes. Se muestra al usuario una lista con las recomendaciones que le han hecho sus amigos.

- Entrada: RD31
- Consulta: RD38

- Salida: RD28

RF20 Canciones mejor valoradas por un usuario. Un usuario solicita una lista de las canciones mejor valoradas proporcionando el identificador de otro usuario.

- Entrada: RD33
- Consulta: RD32
- Salida: RD39

RF21 Eliminar un álbum existente. Un usuario artista proporciona el identificador de uno de sus álbumes para eliminarlo

- Entrada: RD40
- Consulta: RD41

2.3. Validación cruzada de requisitos

RF	Entrada	Manejo	Salida
RF1	RD1	RD2,RD3,RD4	RD14
RF2	RD6	RD4	RD7
RF3	RD8	RD4	RD5
RF4	RD5	RD4	
RF5	RD36	RD4	
RF6	RD5	RD9	RD9
RF7	RD10	RD2	
RF8	RD11	RD2,RD3	RD12
RF9	RD13	RD12	RD15
RF10	RD16	RD3,RD10	RD17
RF11	RD18	RD2,RD3,RD10	
RF12	RD19	RD20	
RF13	RD21	RD20	
RF14	RD22	RD20	
RF15	RD23	RD20	
RF16	RD24	RD25,RD26	
RF17	RD27	RD29	
RF18	RD25	RD37	RD30
RF19	RD31	RD38	RD28
RF20	RD33	RD32	RD39
RF21	RD40	RD41	

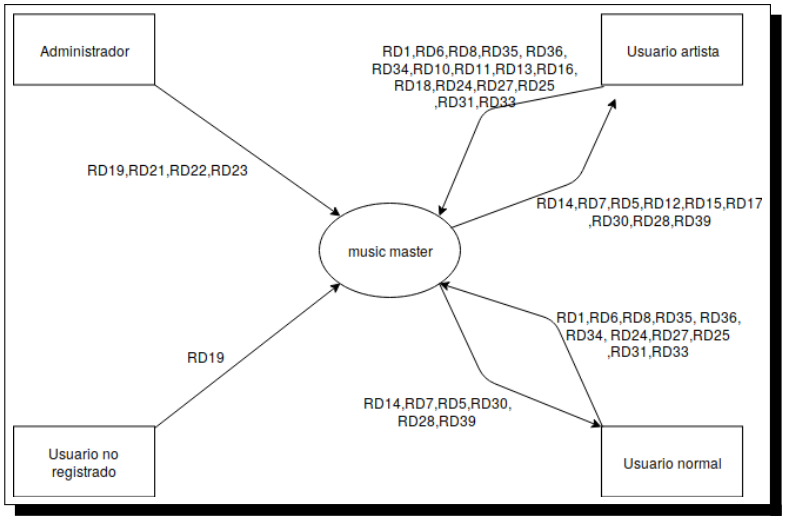
Cuadro 1: Requisitos funcionales

RD	Entrada	Manejo	Salida
RD1	RF1		
RD2		RF1,RF2,RF7,RF8,RF11	
RD3		RF1,RF2,RF8,RF10,RF11	
RD4		RF1,RF2,RF3,RF4,RF5	
RD5			RF3
RD6	RF2		
RD7			RF2
RD8	RF3		
RD9		RF6	
RD10	RF7		
RD11	RF8		
RD12		RF9	RF8
RD13	RF9		
RD14			RF1
RD15			RF9
RD16	RF10		
RD17			RF10
RD18	RF11		
RD19	RF12		
RD20		RF12,RF13,RF14,RF15	
RD21	RF13		
RD22	RF14		
RD23	RF15		
RD24	RF16		
RD25	RF18		
RD26		RF16	
RD27	RF17		
RD28			RF19
RD29		RF17	
RD30			RF18
RD31	RF19		
RD32		RF20	
RD33	RF20		
RD34	RF6		
RD35	RF4		
RD36	RF5		
RD37		RF18	
RD38		RF19	
RD39			RF20
RD40	RF21		
RD41		RF21	

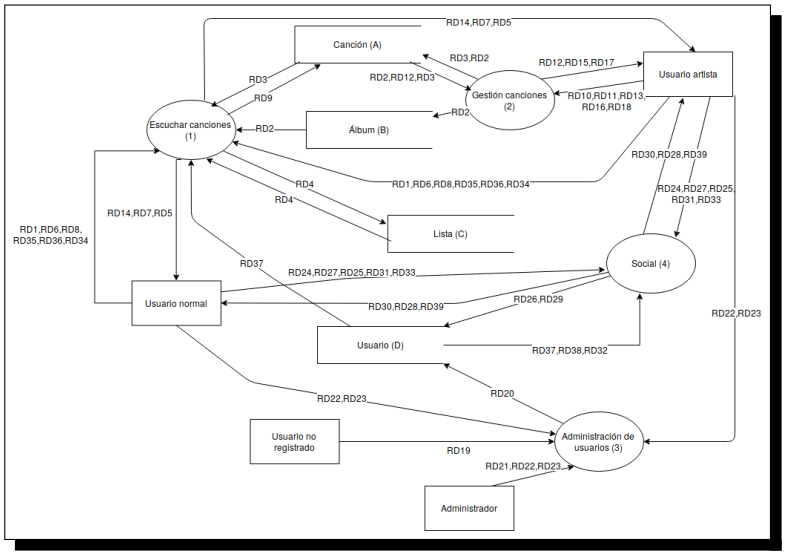
Cuadro 2: Requisitos de datos

3. Diseño.

3.1. Esquema de caja negra.

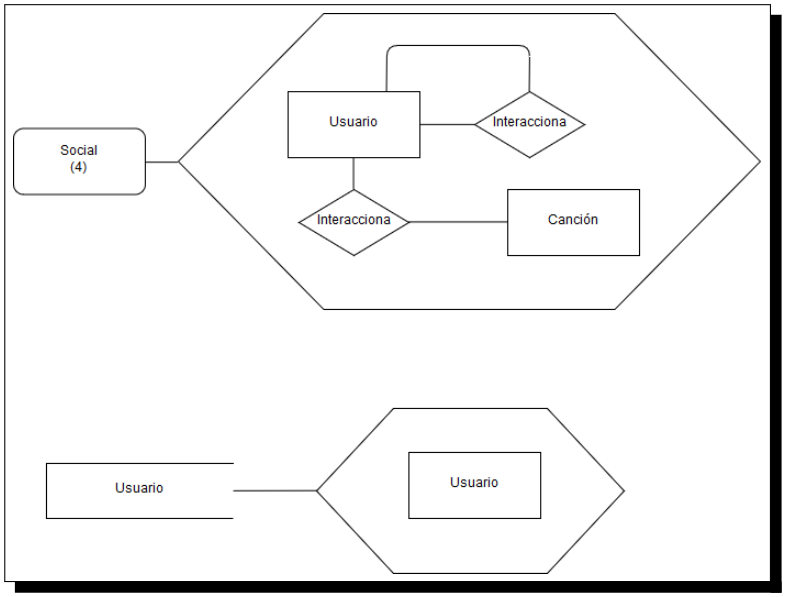


3.2. Esquema armazón DFD0.



3.3. Esquemas externos del DFD0.

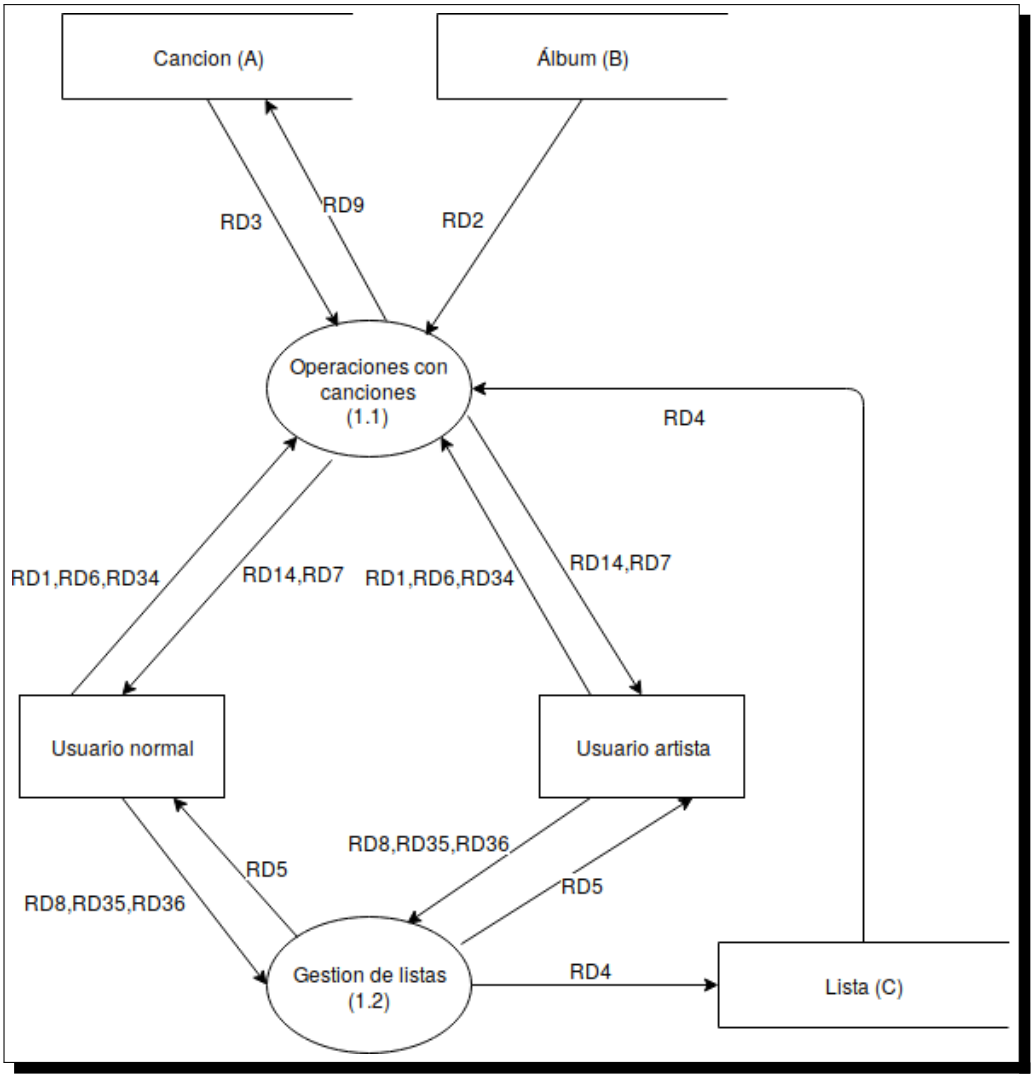
3.3.1. EE0 Módulo social



3.4. Refinamientos.

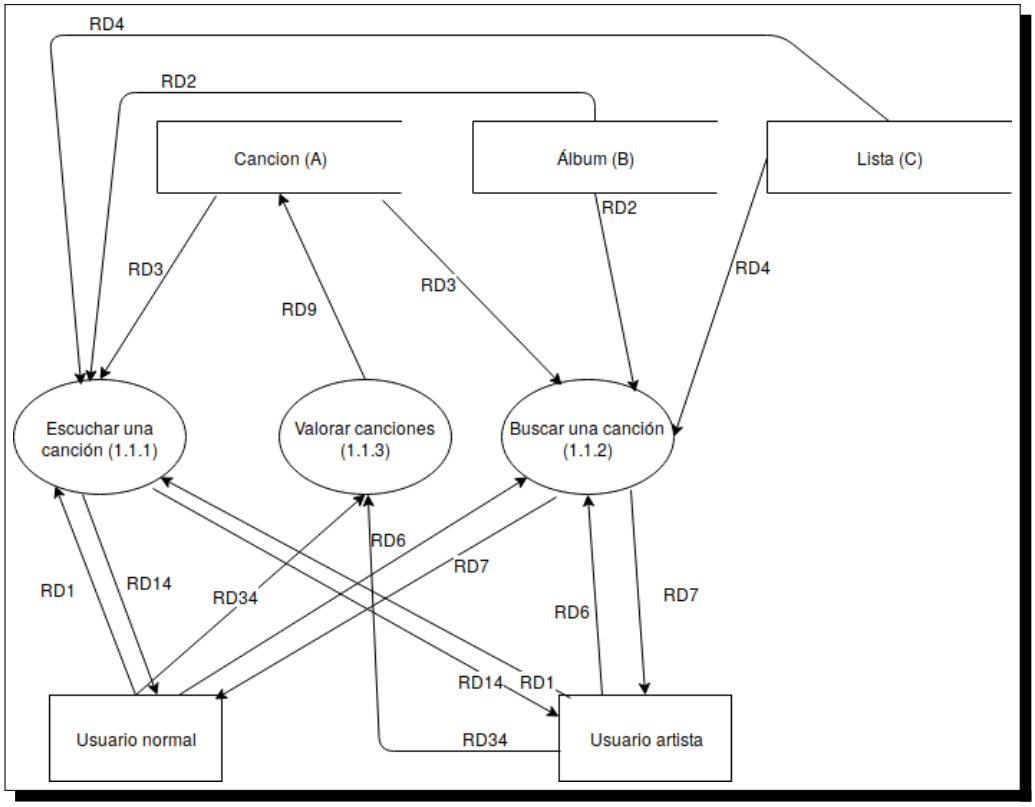
3.4.1. Módulo escuchar canciones.

3.4.1.1. DFD1

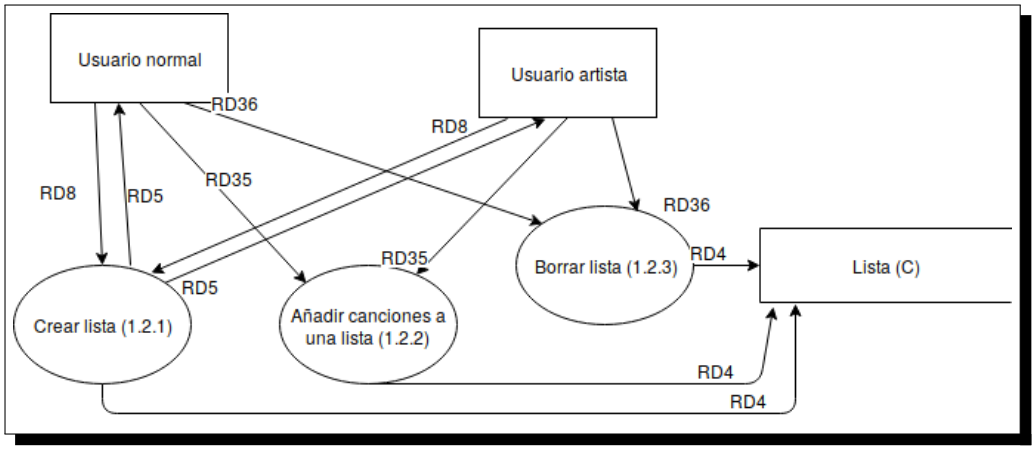


3.4.1.2. DFD2

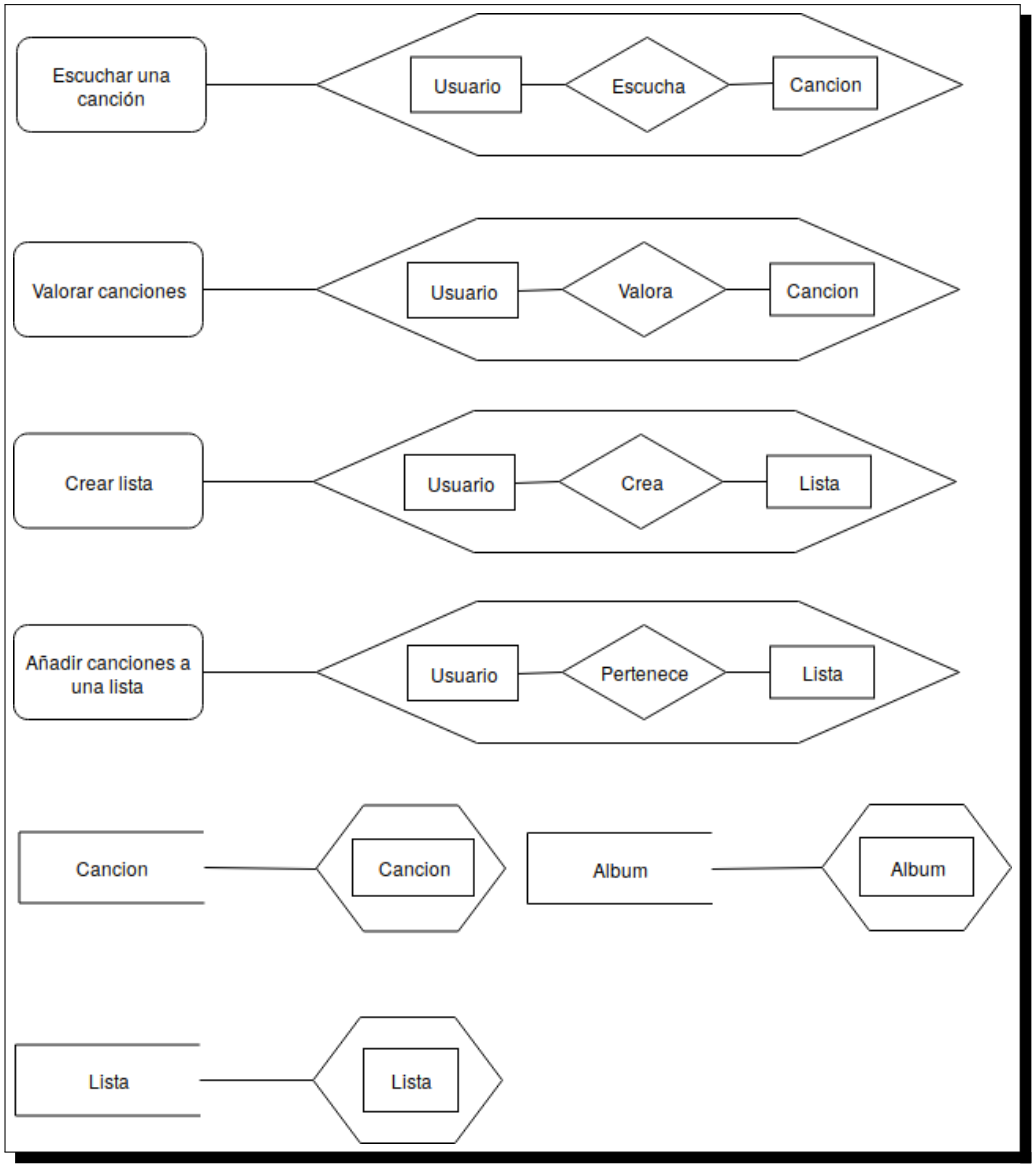
3.4.1.2.1. Primer plano de refinamiento.



3.4.1.2.2. Segundo plano de refinamiento.

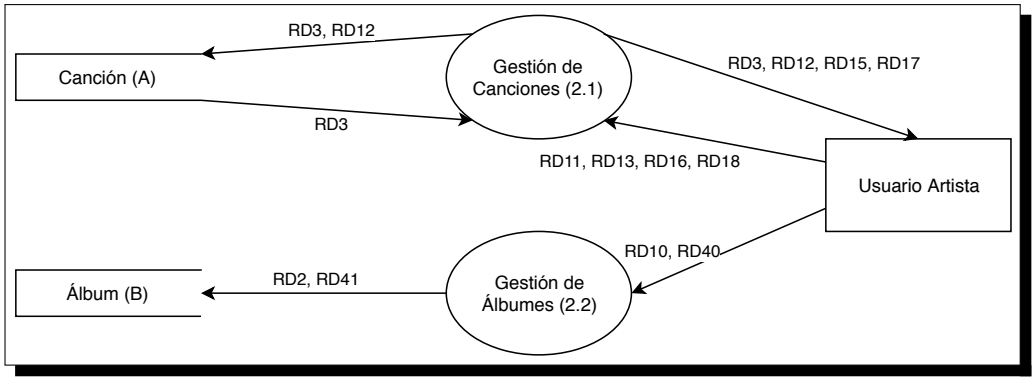


3.4.1.3. Esquema externo.

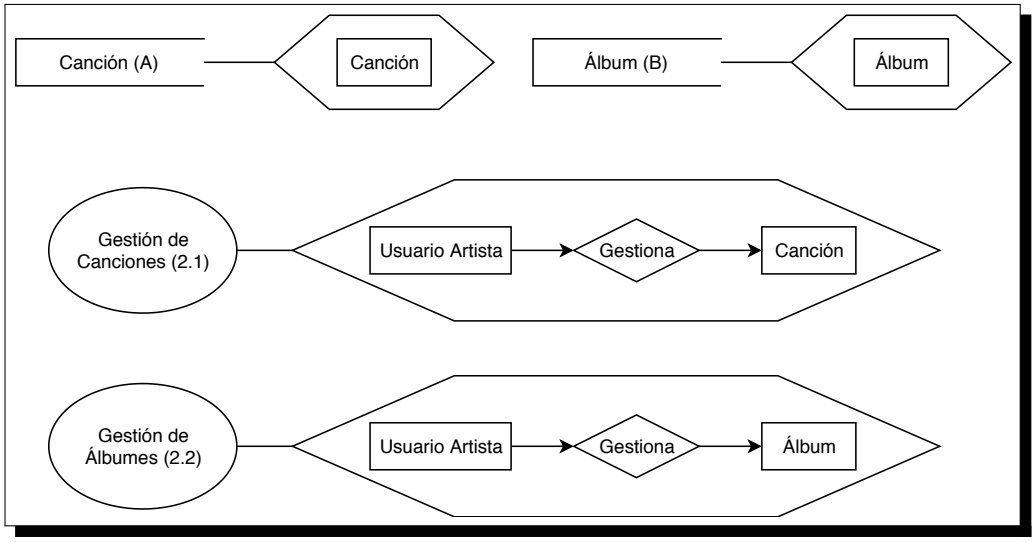


3.4.2. Módulo gestionar canciones.

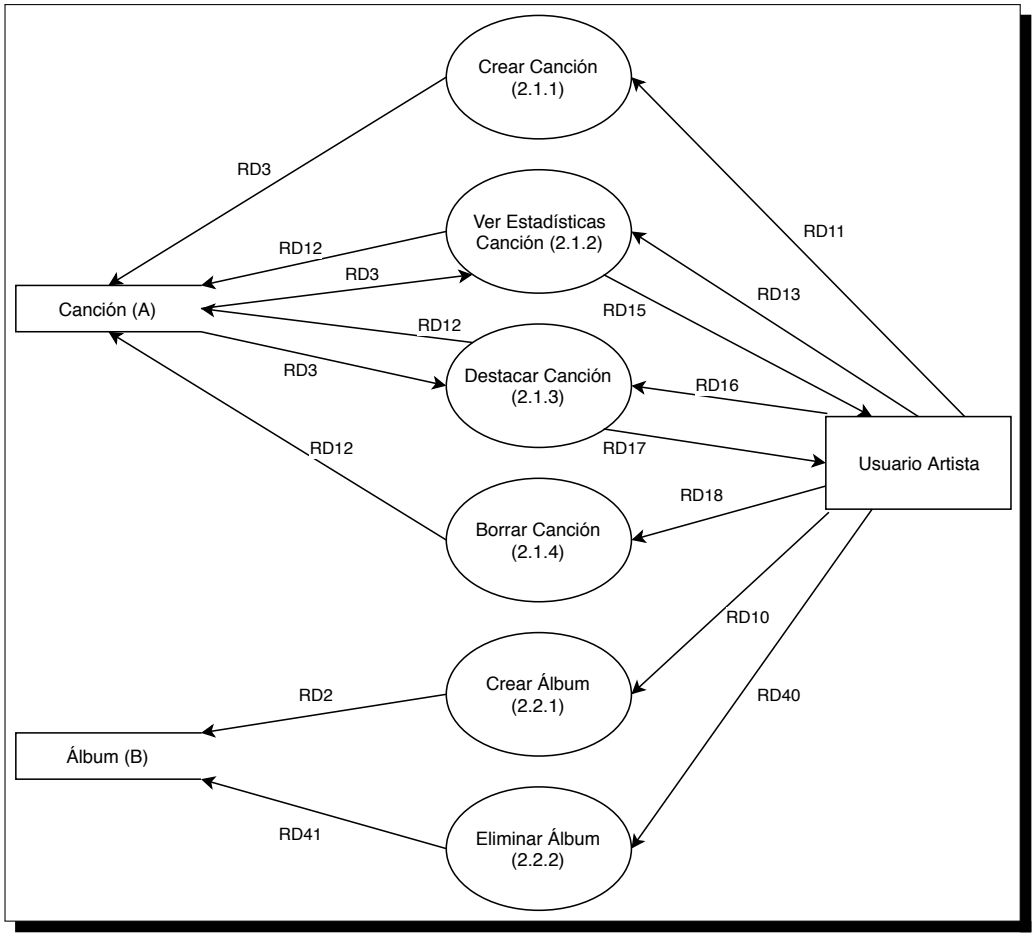
3.4.2.1. DFD1



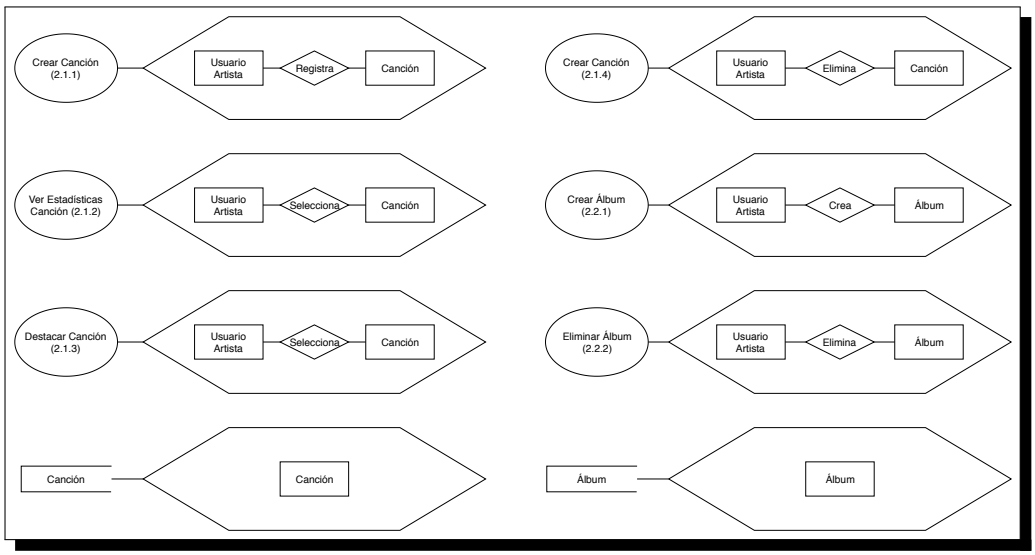
3.4.2.2. EE1



3.4.2.3. DFD2

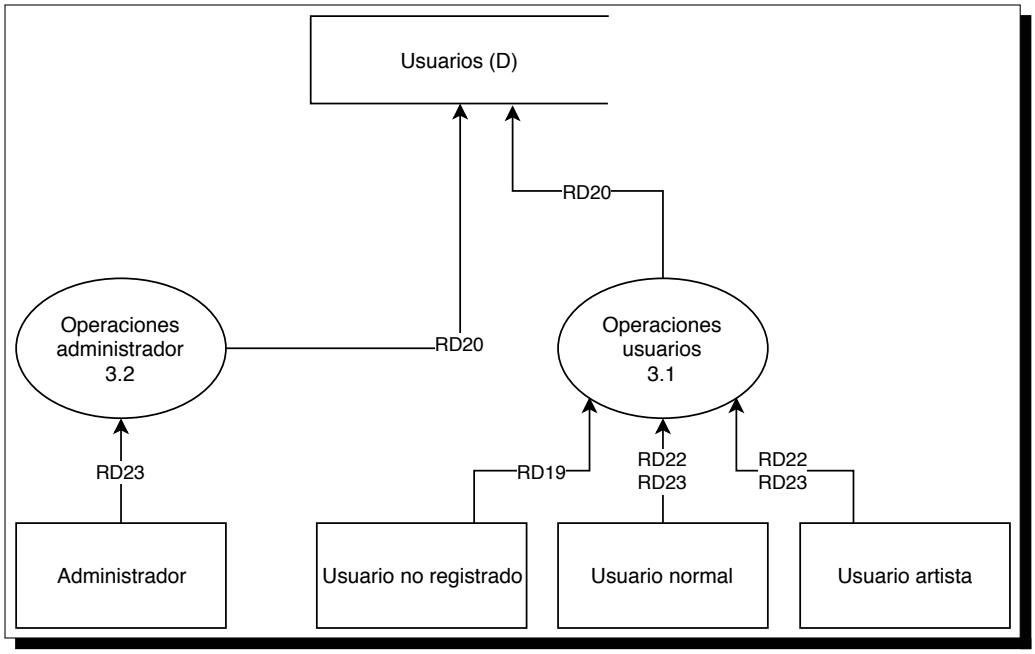


3.4.2.4. Esquema externo.

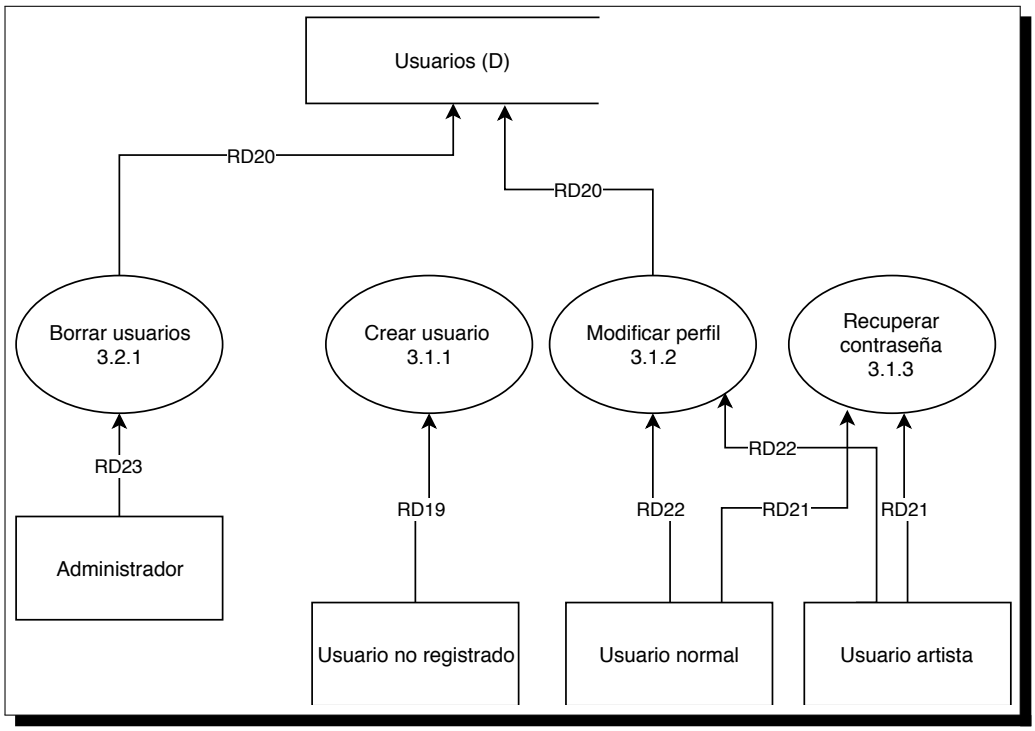


3.4.3. Módulo gestión de usuarios.

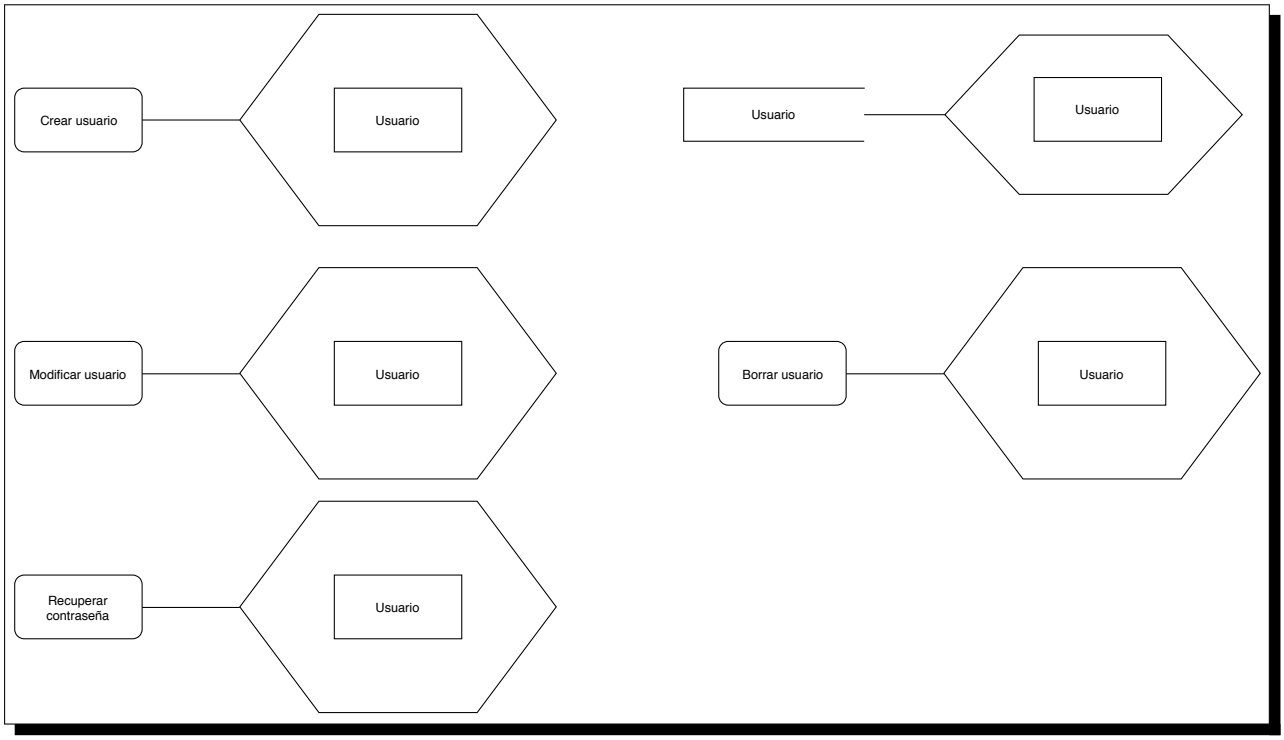
3.4.3.1. DFD1



3.4.3.2. DFD2

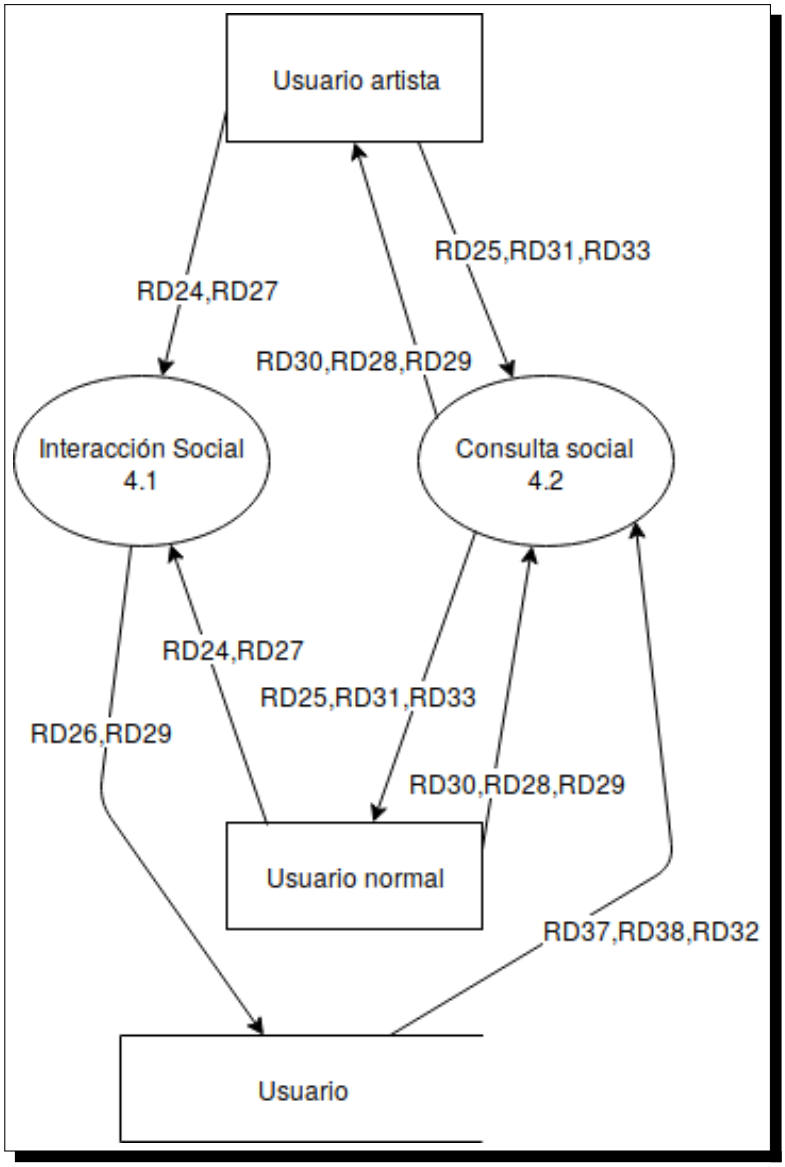


3.4.3.3. Esquema externo.

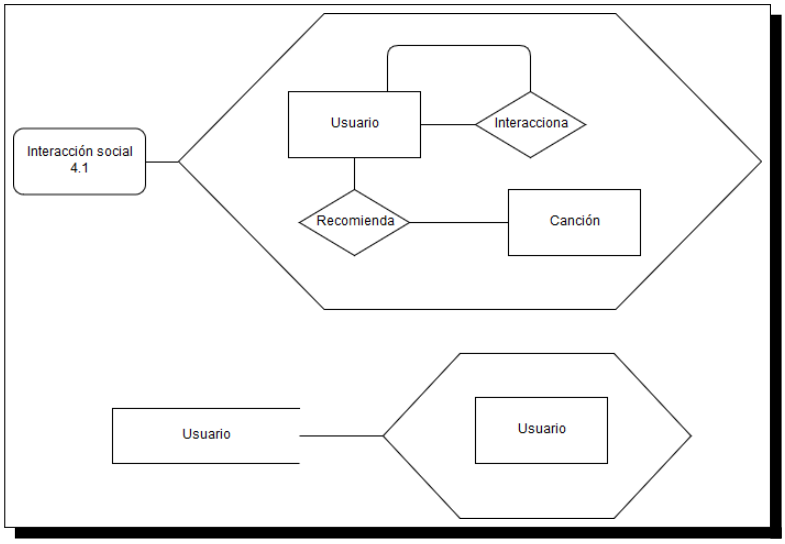


3.4.4. Módulo social.

3.4.4.1. DFD1

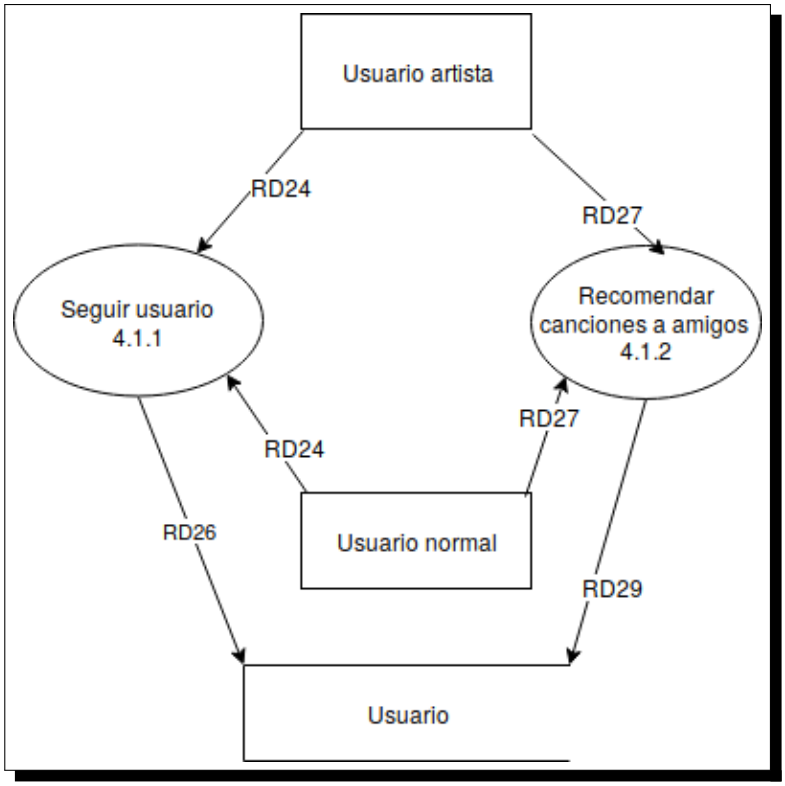

Usuario artista
RD24,RD27
RD25,RD31,RD33
RD30,RD28,RD29
Interacción Social
4.1
Consulta social
4.2
RD24,RD27
RD25,RD31,RD33
RD26,RD29
RD30,RD28,RD29
Usuario normal
RD37,RD38,RD32
Usuario

3.4.4.2. EE1

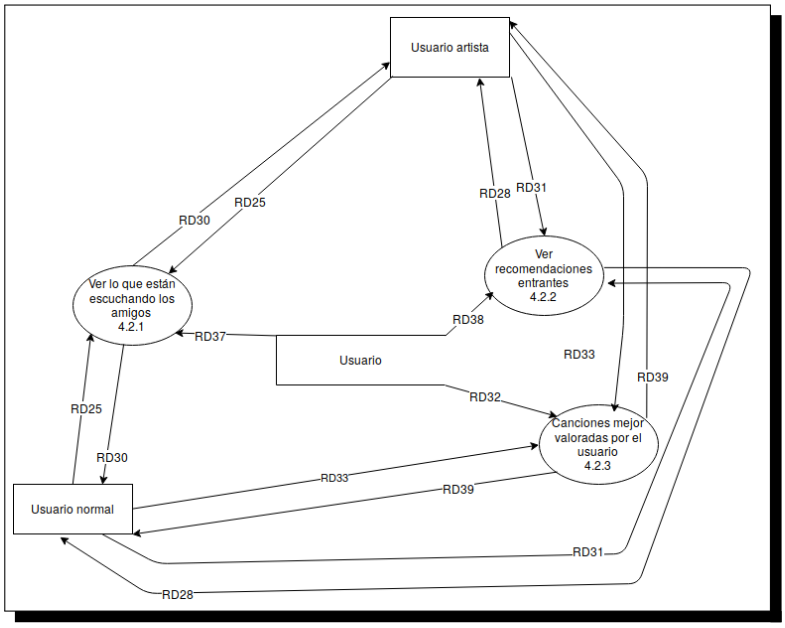


3.4.4.3. DFD2

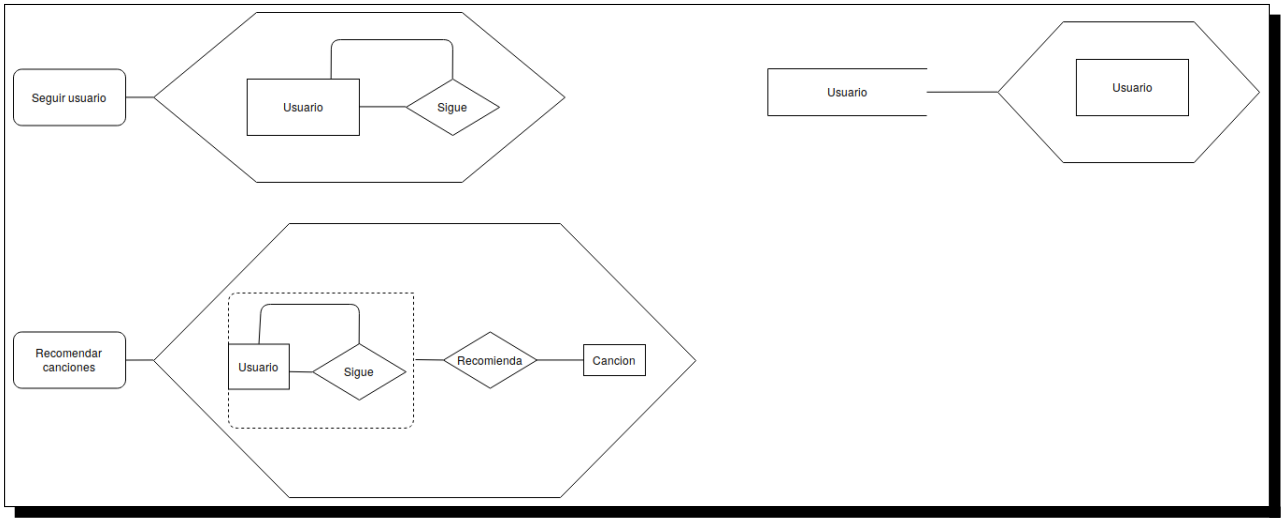
3.4.4.3.1. Primer plano de refinamiento.



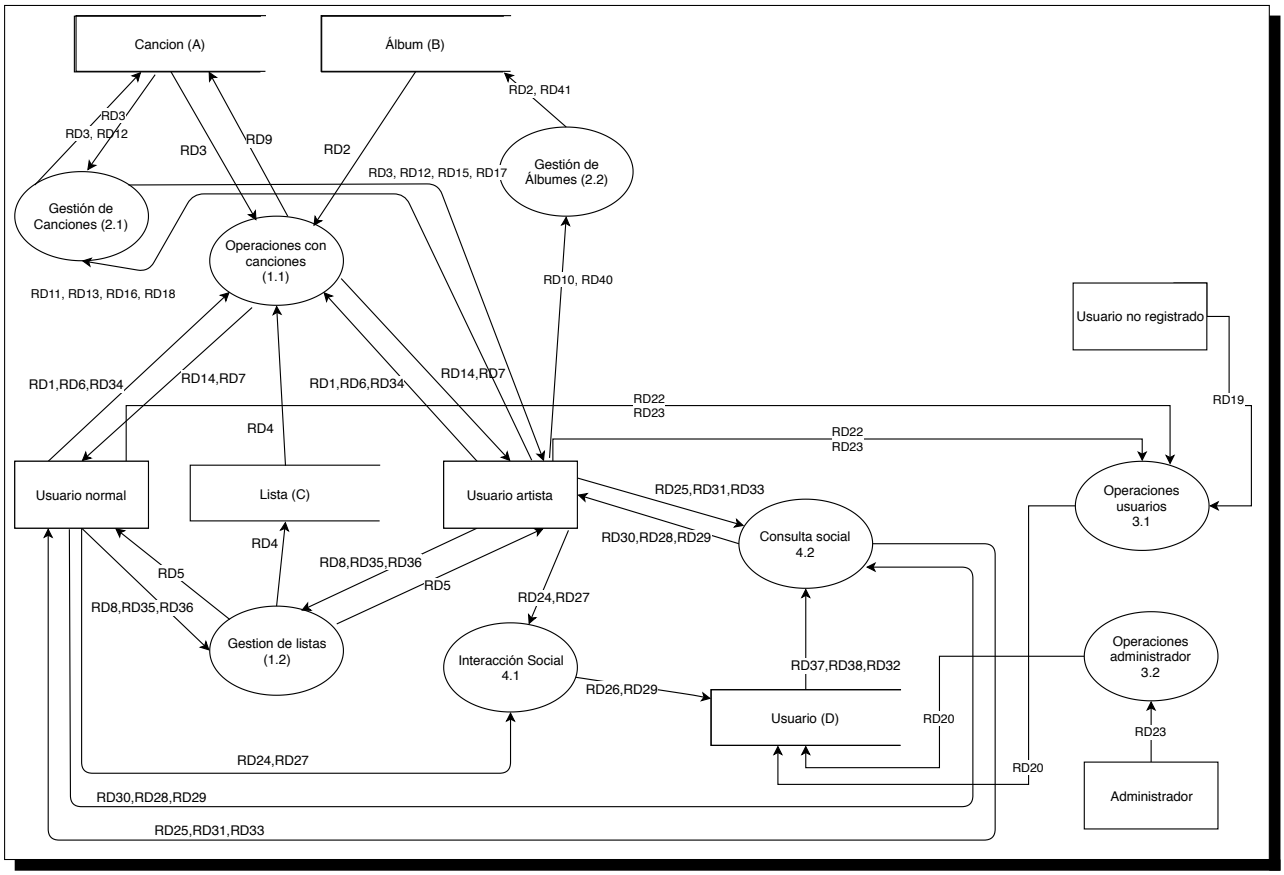
3.4.4.3.2. Segundo plano de refinamiento.



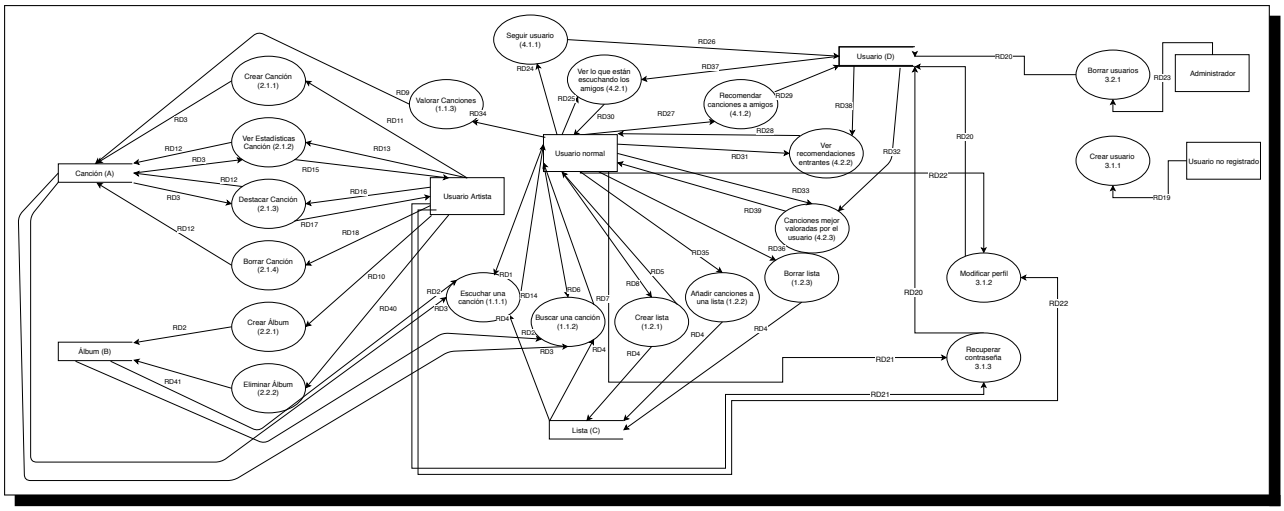
3.4.4.4. Esquema externo.



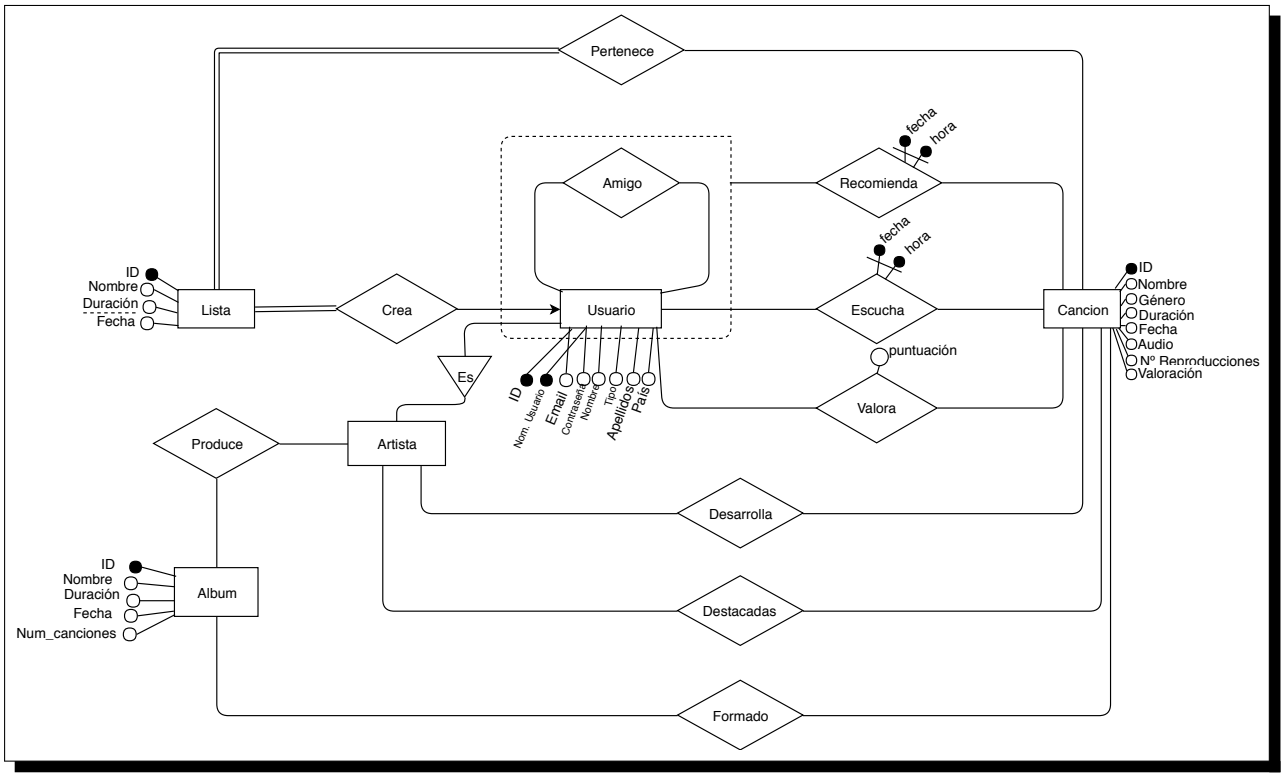
3.4.5. Unión de módulos DFD1.



3.4.6. Unión de módulos DFD2.



3.5. Esquema entidad relación.



3.6. Paso a tablas.

Tabla Usuarios	
Clave primaria	ID usuario
Resto de atributos	nombre usuario, nombre artístico, email, contraseña, nombre, apellidos, tipo, pais.

Tabla Canción	
Clave primaria	ID canción
Resto de atributos	nombre, genero, duración, fecha, ruta audio, numero de reproducciones, valoración.

Tabla Álbum	
Clave primaria	ID album
Resto de atributos	nombre, fecha, numero de canciones, duración

Tabla Lista	
Clave primaria	ID lista
Resto de atributos	nombre, fecha, duración.

Tabla Amigo	
Clave primaria	ID usuario1, ID usuario2.
Ambas son claves externas a ID usuario de la tabla usuario	

Tabla Recomienda	
Clave primaria	ID usuario1, ID usuario2, ID cancion, fecha.
ID usuario1 e ID usuario2 como claves externas a la tabla amigo. ID cancion como clave externa a la tabla cancion.	

Tabla Escucha	
Clave primaria	ID cancion, ID usuario, fecha
ID usuario como clave externa a la tabla usuario. ID canción como clave externa a la tabla canción.	

Tabla Valora	
Clave primaria	ID usuario, ID cancion
ID usuario como clave externa a la tabla usuario. ID canción como clave externa a la tabla canción.	
Resto de atributos	puntuación

Tabla Pertenece	
Clave primaria	ID lista, ID cancion
ID lista como clave externa a la tabla lista. ID canción como clave externa a la tabla canción.	

Tabla Crea	
Clave primaria	ID lista, ID usuario
ID lista como clave externa a la tabla lista. ID usuario como clave externa a la tabla usuario.	

Tabla Formado	
Clave primaria	ID álbum, ID cancion
ID álbum como clave externa a la tabla álbum. ID canción como clave externa a la tabla canción.	

Tabla Desarrolla	
Clave primaria	ID usuario, ID cancion
ID usuario como clave externa a la tabla usuario. ID canción como clave externa a la tabla canción.	

Tabla Produce	
Clave primaria	ID álbum, ID usuario
ID álbum como clave externa a la tabla álbum. ID usuario como clave externa a la tabla usuario.	

Tabla Destacadas	
Clave primaria	ID usuario, ID canción
ID usuario como clave externa a la tabla usuario. ID canción como clave externa a la tabla canción.	

4. Implementación.

4.1. Diseño físico de la base de datos.

4.1.1. Sentencias SQL.

```
create table usuario(
    id_usuario char(8) constraint id_usuario_no_nulo not null
    constraint id_usuario_primaria primary key,
    nombre_usuario varchar2(15)
        constraint nombre_usuario_no_nulo not null
    constraint nombre_usuario_candidata unique,
    email_u varchar2(40) constraint email_no_nulo not null,
    contrasenia_u varchar2(40) constraint contrasenia_no_nula not null,
    nombre_per varchar2(15) constraint nombre_no_nulo not null,
    apellidos_per varchar2(30),
    tipo varchar2(7) constraint tipo_correcto
        check (tipo in ('artista','usuario')),
    pais varchar2(20),
    nombre_artistico varchar2(50));

create table cancion(
    id_cancion char(8) constraint id_cancion_no_nulo not null
    constraint id_cancion_primaria primary key,
    nombre_cancion varchar2(50)
        constraint nombre_cancion_no_nulo not null,
    genero varchar2(20) constraint genero_no_nulo not null,
    duracion_seg number(3) constraint duracion_seg_no_nula not null,
    fecha_c date constraint fecha_cancion_no_nula not null,
    ruta_audio varchar2(100),
    num_repro number(10),
    valoracion number(2,1));

create table lista(
    id_lista char(8) constraint id_lista_no_nulo not null
    constraint id_lista_primaria primary key,
    nombre_lista varchar2(50) constraint nombre_lista_no_nulo not null,
    fecha_l date constraint fecha_lista_no_nula not null,
    duracion_seg number(6));

create table album(
    id_album char(8) constraint id_album_no_nulo not null
    constraint id_album_primaria primary key,
    nombre_album varchar2(50)
        constraint nombre_album_no_nulo not null,
    fecha_a date constraint fecha_album_no_nula not null,
    num_canciones number(5),
```

```
    duracion number(6));

create table pertenece(
    id_lista,
    id_cancion,
    constraint pertenece_clave_primaria primary key (id_lista,id_cancion)
    constraint id_lista_ext_pertenece foreign key(id_lista)
        references lista(id_lista),
    constraint id_cancion_ext_pertenece foreign key(id_cancion)
        references cancion(id_cancion));

create table crea(
    id_usuario,
    id_lista,
    constraint crea_clave_primaria primary key (id_usuario,id_lista),
    constraint id_usuario_ext_crea foreign key(id_usuario)
        references usuario(id_usuario),
    constraint id_lista_ext_crea foreign key(id_lista)
        references lista(id_lista));

CREATE TABLE amigo
  (
    id_usuario1,
    id_usuario2,
    CONSTRAINT clave_primaria PRIMARY KEY (id_usuario1,id_usuario2),
    CONSTRAINT id_usuario1_ext_pertenece FOREIGN KEY(id_usuario1)
    REFERENCES usuario(id_usuario),
    CONSTRAINT id_usuario2_ext_pertenece FOREIGN KEY(id_usuario2)
    REFERENCES usuario(id_usuario)
  );

create table recomienda(
  id_usuario1,
  id_usuario2,
  id_cancion,
  fecha date not null,
  constraint clave_pri
        primary key (id_usuario1,id_usuario2,id_cancion,fecha),
  foreign key (id_usuario1,id_usuario2)
        references amigo(id_usuario1,id_usuario2),
  foreign key (id_cancion) references cancion(id_cancion)
);
```

```
create table valora(
  id_usuario,
  id_cancion,
  puntuacion number(3,1),
  primary key (id_usuario,id_cancion),
  foreign key (id_usuario) references usuario(id_usuario),
  foreign key (id_cancion) references cancion(id_cancion)
);

create table escucha(
  id_usuario,
  id_cancion,
  fecha date not null,
  primary key (id_usuario,id_cancion,fecha),
  foreign key (id_usuario) references usuario(id_usuario),
  foreign key (id_cancion) references cancion(id_cancion)
);

create table formado(
  id_album,
  id_cancion,
  primary key (id_album,id_cancion),
  foreign key (id_album) references album(id_album),
  foreign key (id_cancion) references cancion(id_cancion)
);

create table desarrolla(
  id_usuario,
  id_cancion,
  primary key (id_usuario,id_cancion),
  foreign key (id_usuario) references usuario(id_usuario),
  foreign key (id_cancion) references cancion(id_cancion)
);

create table produce(
  id_usuario,
  id_album,
  primary key (id_usuario,id_cancion),
  foreign key (id_usuario) references usuario(id_usuario),
  foreign key (id_album)   references album(id_album)
);

create table destacadas(
```

```
  id_usuario,
  id_cancion,
  primary key (id_usuario,id_cancion),
  foreign key (id_usuario) references usuario(id_usuario),
  foreign key (id_cancion) references cancion(id_cancion)
);
```

4.2. Implementación modulo escuchar música.

4.2.1. Implementación del disparador.

El disparador se encarga de controlar la inserción de una tupla en la tabla pertenece. Esta tabla es la encargada de guardar las canciones que pertenecen a una lista determinada. El trabajo del disparador es que cuando una canción pertenezca a mas de dos lista diferentes muestra un mensaje

El funcionamiento consiste en contar cuantas veces aparece la misma canción en la tabla pertenece (cada vez que aparezca quiere decir que esta en otra lista) antes de la inserción de esta a una nueva lista. Si es mayor que uno muestra el mensaje. Muestra el mensaje cuando es mayor que uno y no dos por que el disparador es un BEFORE INSERT y a ese uno hay que sumarle la inserción próxima.

```
CREATE OR REPLACE TRIGGER pedro_trigg
BEFORE INSERT ON pertenece
FOR EACH ROW
DECLARE
num_canciones number := 0;
BEGIN
    SELECT count(id_cancion)
    INTO num_canciones
    FROM pertenece
    where id_cancion = :new.id_cancion;

        if (num_canciones > 1) THEN
                DBMS_OUTPUT.PUT_LINE('La cancion pertenece a mas de dos l
        END IF;
END;
```

4.2.2. Implementación de añadir canción a una lista.

La implementación ha sido implementada en Java y la interfaz es mediante lineas de comandos. Se muestra un menú y el usuario tiene que elegir una opción.

El acceso a la base de datos se realiza JDBC, por eso la implementación está escrita en Java. Para el desarrollo he usado JDeveloper.

```
package ddsi_practica_final;
import java.sql.*;
import java.sql.SQLException.*;
import oracle.jdbc.driver.*;
```

```
import java.util.Scanner;

public class pedro_aniadir_canciones_a_lista
{
Connection conn;
Statement stmt;
PreparedStatement pstmt;

    public pedro_aniadir_canciones_a_lista() {
        super();
    }
    public void initialize() throws Exception
    {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        conn = DriverManager.getConnection
        ("jdbc:oracle:thin:@oracle0.ugr.es:1521/practbd.oracle0.ugr.es",
      "x7151952","x7151952");

        conn.setAutoCommit (false);

        System.out.println("Conectado con exito");
    }

    public static void main(String[] args)
    throws Exception
    {

        Principal principal = new Principal();
        principal.initialize();

        String menu = "Introduzca una opcion\n" +
                      "1 - Ver listas creadas\n" +
                      "2 - Añadir cancion a una lista\n" +
                      "3 - Crear una lista\n" +
                      "4 - Ver listas con canciones\n" +
                      "0 - Terminar\n";
        String id_lista;

        Scanner reader = new Scanner(System.in);  // Reading from System.in
        System.out.println(menu);

        int opcion = reader.nextInt();

        while(opcion != 0){
            if(opcion == 1){
```

```
            int listaCount = principal.listListas();
            if (listaCount == 0 ){
                System.out.println ("ERROR: No hay listas");
            }
            else{
                System.out.println ("Hay " + listaCount + " listas\n");
            }
        }
        else if(opcion == 2){
            String id_cancion;

            System.out.println("Introduzca ID_LISTA y ID_CANCION:");
            id_lista = reader.next();
            id_cancion = reader.next();

            int result = principal.insertCancion(id_lista,id_cancion);
            if (result == 0){
                System.out.println("-Cancion aniadida con exito\n");
            }
        }
        else if(opcion == 3){
            String nombre_lista;
            String fecha_lista;

            System.out.println("Introduzca ID Nombre Fecha('dd/mm/aa')");
            id_lista = reader.next();
            nombre_lista = reader.next();
            fecha_lista = reader.next();

            int result = principal.insertLista(id_lista,nombre_lista,
                    fecha_lista);
            if (result == 0){
                System.out.println("-Lista aniadida con exito\n");
            }
        }
        else if(opcion == 4){
            int listaCount = principal.listListasCanc();
            if (listaCount == 0 ){
                System.out.println ("ERROR: No hay listas con canciones");
            }
            else{
                System.out.println ("Hay " + listaCount +
                " listas con canciones\n");
            }
        }
```

```
            System.out.println(menu);
            opcion = reader.nextInt();
        }
        reader.close();
    }

    //Metodo para listar listas
    public int listListas() throws Exception
    {
        String sql = "SELECT * FROM LISTA";
        stmt = conn.createStatement ( );
        ResultSet rset = stmt.executeQuery(sql);

        int count = 0;
        while (rset.next ())
        {
        count++;
        System.out.println("ID: " + rset.getString(1) + "\n" +
                           "Nombre: " + rset.getString(2) + "\n" +
                           "Fecha creacion: " + rset.getString(3) + "\n");
        }
        return count;
    }

    //Metodo para listar listas
    public int listListasCanc() throws Exception
    {
        String sql = "SELECT * FROM PERTENECE";
        stmt = conn.createStatement ( );
        ResultSet rset = stmt.executeQuery(sql);

        int count = 0;
        while (rset.next ())
        {
        count++;
        System.out.println("ID Lista: " + rset.getString(1) + "\n" +
                           "ID Cancion: " + rset.getString(2) + "\n");
        }
        return count;
    }

    public int insertLista(String idLista, String nombre, String fecha)
    {
        try
```

```
        {
        pstmt = conn.prepareStatement("INSERT INTO LISTA VALUES(?,?,?,NULL)");
        pstmt.setString(1, idLista);
        pstmt.setString(2, nombre);
        pstmt.setString(3, fecha);
        pstmt.executeUpdate();
        conn.commit();
        return 0; // return OK
        }
        catch (SQLException e)
        {
        System.out.println("Error insertando nueva lista" + e);
        return 1; // return error
        }
    }

    public int insertCancion(String idLista, String idCancion)
    {
        try
        {
        pstmt = conn.prepareStatement("INSERT INTO PERTENECE VALUES(?,?)");
        pstmt.setString(1, idLista);
        pstmt.setString(2, idCancion);
        pstmt.executeUpdate();
        conn.commit();
        return 0; // return OK
        }
        catch (SQLException e)
        {
        System.out.println("Error aniadiendo cancion a una lista" + e);
        return 1; // return error
        }
    }
}
```

4.3. Implementación modulo social.

4.3.1. Implementación del disparador.

Este disparador controla la inserción de una tupla en la tabla recomendación. Es decir comprueba una condición cuando un usuario recomienda una canción a otro usuario. Dicha condición consiste en que un usuario no puede recomendar una canción si no la ha escuchado previamente.

Básicamente, el funcionamiento consiste en comprobar la tabla escucha (donde reside el registro de reproducciones que han realizado los usuarios) y mirar si el usuario que recomienda ha reproducido la canción que se recomienda canción. Si no encuentra ningún resultado, levanta una excepción.

```
create or replace TRIGGER dario_trigg
BEFORE INSERT ON recomienda
FOR EACH ROW
DECLARE
no_escuchada EXCEPTION;
existe number;
BEGIN
SELECT count(*) INTO existe FROM escucha
WHERE id_cancion = :new.id_cancion and id_usuario = :new.id_usuario1;
    IF (existe = 0) THEN
            RAISE no_escuchada;
    END IF;
    EXCEPTION
    WHEN no_escuchada THEN
    RAISE_APPLICATION_ERROR(-20000,'No has escuchado la cancion');
END;
```

4.3.2. Implementación de un proceso.

El proceso acordado para la implementación consiste mostrar que están escuchando mis amigos. Debido a la dificultad de mostrar lo que se está reproduciendo en un instante, se decidió mostrar las reproducciones realizadas “hoy”, es decir, en un día completo.

El proceso ha sido implementado en Python y la interfaz es línea de comandos (CLI), debido a la falta de tiempo y el poco conocimiento acerca de interfaces gráficas.

La conexión a la base de datos se realiza mediante el paquete “cx_Oracle”, que nos permite realizar las consultas y manejar los datos consultados.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import os
import cx_Oracle


def menu():
  print ("Opciones disponibles:")
  print ("\t1 - Ver que han escuchado mis amigos hoy.")
  print ("\t0 - Salir")

def mostrar_info_amigo(info_amigo):
  for row in info_amigo:
    print "Usuario: " + row[0]
    print "Nombre: " + row[1] + " " + row[2]
  print "-----------------------------------------------"

#############################################################

conn_str = 'USUARIO/PASSWORD@HOSTNAME:PORT/SERVICENAME'
conn = cx_Oracle.connect(conn_str)

my_id = 'UN000002' # Hipotético usuario logueado en el sistema
hoy = '17/09/2017' # Hipotécica fecha actual

c = conn.cursor()

while True:
  menu()

  opcionMenu = str(input("Selecciona una acción: "))

  if opcionMenu == '1':
  #Listamos los amigos
    c.execute("SELECT id_usuario2 FROM amigo
                  WHERE id_usuario1 = '%s'" %(my_id))
    id_amigos = c.fetchall()

    contador = 1
    info_amigo = []
    #MOSTRAR LISTA DE AMIGOS
    print "-----------------------------------------------"
    for ID in id_amigos:
```

```
    c.execute("SELECT nombre_usuario, nombre_per, apellidos_per
                  FROM usuario WHERE id_usuario = '%s'" %(ID))
    info_amigo = c.fetchall()
    print "Amigo " + str(contador)
    mostrar_info_amigo(info_amigo)
    contador += 1

  opcionMenu = str(input("Selecciona un amigo: "))
  num_usuario = int(opcionMenu)-1

  #RECUPERAR ID AMIGO SELECCIONADO
  c.execute("SELECT id_usuario2 FROM amigo
                WHERE id_usuario1 = '%s'" %(my_id))
  id_amigos = c.fetchall()
  amigo = id_amigos[num_usuario]

  #Recuperamos nombre de amigo
  c.execute("SELECT nombre_usuario FROM usuario
                WHERE id_usuario = '%s'" %(amigo[0]))
  nombre_amigo = c.fetchone()

  #Consultamos las canciones escuchadas hoy por el amigo
  #seleccionado
  c.execute("SELECT * FROM escucha WHERE id_usuario = '%s'
      and fecha = to_date('%s','DD/MM/YYYY')" %(amigo[0], hoy))
  canciones = c.fetchall()


  #Mostramos lista de canciones escuchadas por el amigo
  #seleccionado
  print "\n-------------------------------------------------"
  print "Lista de canciones escuchadas hoy por %s:"
                                              %(nombre_amigo)
  for cancion in canciones:
    c.execute("SELECT nombre_cancion FROM cancion
              WHERE id_cancion = '%s'" %(cancion[1]))
    nombre_cancion = c.fetchone()[0]
    print "- " + nombre_cancion

  print "\n\n"


if opcionMenu == '0':
```

```
    conn.close()
    break
```

4.4. Valoración de canciones.

4.4.1. Implementación del disparador.

El disparador que tenemos que hacer tiene que comprobar que cuando vamos a valorar una canción, ésta ha sido escuchada al menos una vez por el usuario que valora. Para hacer esto implementamos un disparador que se ejecuta antes de cada operación `INSERT` en la tabla `valora`.

Este disparador lo que hace es comprobar el número de escuchas que ha tenido la canción a valorar por el usuario que valora. Si este número es 0 (no ha escuchado nunca la canción) lanza una excepción con un mensaje que avisa al usuario que no puede ejecutarse esta operación.

El código del disparador implementado es el siguiente:

```
create or replace TRIGGER javier_trigg
BEFORE INSERT ON valora
FOR EACH ROW
DECLARE
no_escuchada EXCEPTION;
existe number;
BEGIN
SELECT count(*) INTO existe FROM escucha WHERE
      id_cancion = :new.id_cancion and id_usuario = :new.id_usuario;
    IF (existe = 0) THEN
          RAISE no_escuchada;
    END IF;
    EXCEPTION
    WHEN no_escuchada THEN
    RAISE_APPLICATION_ERROR(-20001,'No puedes valorar una cancion que
                            no has escuchado');
END;
```

Para probar que este disparador funciona correctamente hacemos las siguiente operaciones:

- Vamos a intentar crear una valoración de una canción que no hemos escuchado

```
INSERT INTO valora(id_usuario, id_cancion, puntuacion)
values('UN000003', 'C0000001', 5.0);
```

- Nos da un mensaje de error, ya que no hemos escuchado la canción (el disparador genera una excepción)

```
INSERT INTO valora(id_usuario, id_cancion, puntuacion)
```

```
values('UN000003', 'C0000001', 5)
Informe de error -
ORA-20001: No puedes valorar una cancion que no has escuchado
ORA-06512: en "X7151952.JAVIER_TRIGG", línea 11
ORA-04088: error durante la ejecución del disparador 'X7151952.JAVIER_TRIGG'
```

- Insertamos ahora una escucha por el usuario que desea valorar

```
INSERT INTO escucha(id_usuario, id_cancion, fecha)
values('UN000003', 'C0000001', to_date('19/01/2019','DD/MM/YYYY'));
```

- Si insertamos ahora una valoración funciona sin problemas.

```
1 fila insertadas
```

- Devolvemos la BD a su estado original

```
DELETE FROM escucha where id_usuario = 'UN000003' and ID_CANCION = 'C0000001';
DELETE FROM valora where id_usuario = 'UN000003' and ID_CANCION = 'C0000001';
commit;
```

4.4.2. Implementación de un proceso.

El proceso a implementar según lo acordado con el profesor es el de valorar canciones. Para ello vamos a desarrollar una aplicación que muestre las canciones del sistema y permita valorarlas. Si la canción no ha sido escuchada por el usuario da un error advirtiendo al usuario que no puede valorar una canción que no ha escuchado. Además muestra un error si la canción ya ha sido valorada por el usuario.

El proceso ha sido implementado en Python y Javascript. Se ha diseñado una interfaz gráfica basada en tecnologías web que hace consultas a una API creada en Python con el microframework Flask. La conexión a la base de datos se realiza mediante el paquete "cx_Oracle", que nos permite realizar las consultas y manejar los datos consultados.

El código del servidor es el siguiente:

```
#!flask/bin/python
from flask import Flask, jsonify, request

import cx_Oracle
import datetime
```

```
# Conexión a la BD
conn_str = 'USUARIO/PASSWORD@HOSTNAME:PORT/SERVICENAME'
conn_str = 'x7151952/x7151952@oracle0.ugr.es:1521/practbd.oracle0.ugr.es'
conn = cx_Oracle.connect(conn_str)

my_id = 'UN000003' # Hipotético usuario logueado en el sistema
hoy = '17/09/2017' # Hipotécica fecha actual
hoy = datetime.date.today().strftime("%d/%m/%Y")

c = conn.cursor()

app = Flask(__name__,static_url_path='', static_folder='')

canciones = [
 {
  'id':'ID',
  'nombre': 'Hola',
  'valoracion': 5,
  'escuchada': 0
 }
]

@app.route('/')
def root():
 return app.send_static_file('index.html')

@app.route('/music_master/api/v1.0/canciones', methods=['GET'])
def get_canciones():
 c.execute("SELECT id_cancion, nombre_cancion, valoracion FROM cancion")
 query_canciones = c.fetchall()
 canciones = []
 for cancion in query_canciones:
  d = {}
  d['id'] = cancion[0]
  d['nombre'] = cancion[1]
  d['valoracion'] = str(int(int(cancion[2])/2))
  consulta = "SELECT count(*) FROM escucha WHERE \
  (id_cancion = '"+ cancion[0]+ "' and id_usuario = '"+my_id+"')"
  print(consulta)
  c.execute(consulta)
  cuenta = c.fetchall()
  print(cuenta)
  d['escuchada'] = int(cuenta[0][0])
```

```
    canciones.append(d)
  print(canciones)
  return jsonify({'canciones': canciones})


# Mando por POST un objeto en un json [id]
@app.route('/music_master/api/v1.0/escuchar', methods=['POST'])
def escuchar_cancion():
  if not request.json or not 'id' in request.json:
    abort(400)
  id = request.json['id']
  estado=True
  try:
    consulta = "INSERT INTO escucha(id_usuario, id_cancion, fecha) \
    values('"+my_id+"', '"+id+"', to_date('"+hoy+"','DD/MM/YYYY'))"
    c.execute(consulta)
  except cx_Oracle.DatabaseError as e:
    error, = e.args
    estado = False

  return jsonify({'resuelto': estado})

# Mando por POST dos objetos en un json [id, valoracion]
@app.route('/music_master/api/v1.0/valorar', methods=['POST'])
def valorar_cancion():
  id = request.json['id']
  valoracion = request.json['valoracion']
  estado=200
  try:
    consulta = "INSERT INTO valora(id_usuario, id_cancion, puntuacion)\
     values('"+my_id+"', '"+id+"', "+str(int(valoracion)*2.0)+")"
    c.execute(consulta)
    conn.commit()
  except cx_Oracle.DatabaseError as e:
    error, = e.args
    if error.code == 20001:
      estado = 20001
    else:
      estado = 500
    print(error)


  return jsonify({'estado': estado})
```

```
if __name__ == '__main__':
 app.run(debug=True)
 c.execute("SELECT * FROM cancion")
 canciones_prueba = c.fetchall()
 print(canciones_prueba)
 conn.close()
```

El código Javascript que consume la API es el siguiente:

```
var canciones

function get_canciones() {
 //document.getElementById("demo").innerHTML = "Hello World!";
 var misCabeceras = new Headers();

 var miInit = { method: 'GET',
      headers: misCabeceras,
      mode: 'no-cors',
      cache: 'default' };
 fetch('http://localhost:5000/music_master/api/v1.0/canciones',miInit)
 .then(function(response) {
  return response.json();
 })
 .then(function(myJson) {
  console.log(myJson);
  canciones = myJson['canciones']
  $('#lista_canciones').empty();
  for( var i = 0; i < canciones.length; i++ )
  {
   o = canciones[i];
   var html = '';
   html='<li class="list-group-item">'+o['nombre']+'</li>';
   html='<button type="button" class="list-group-item list-group-item-act
   if (o['escuchada']!=0) {
    html+='<span class="badge badge-primary badge-pill">E</span>'
   }

   html+='</button>';
   $('#lista_canciones').append(html);
  }
 });
  }
```

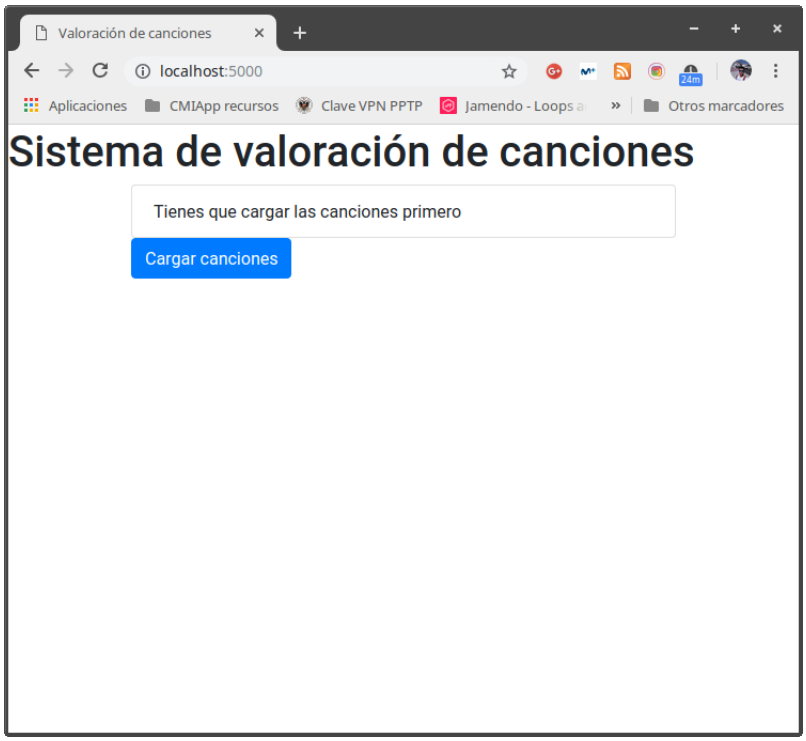
```
  $('#cargar_canciones').on('click', function (e) {

 get_canciones();

})

function open_modal(id) {
 var cancion;
 for( var i = 0; i < canciones.length; i++ )
  {
   if (canciones[i]['id']==id) {
    cancion = canciones[i];
   }
  }
 $('#titulo_cancion').text(cancion['nombre']);
 $('#id_cancion').text(cancion['id']);
 $("#stars").attr("data-rating", cancion['valoracion']);
 $('#myModal').modal('toggle')
}

$(function() {
 return $(".starrr").starrr();
  });

  $( document ).ready(function() {

 $('#stars').on('starrr:change', function(e, value){
   $('#count').html(value);
   post_valoracion();
 });
  });


function post_valoracion() {
 var url = 'http://localhost:5000/music_master/api/v1.0/valorar';
 var data = { id: $('#id_cancion').html(), valoracion: $('#count').html()

 fetch(url, {
  method: 'POST', // or 'PUT'
  body: JSON.stringify(data), // data can be `string` or {object}!
  headers: {
   'Content-Type': 'application/json'
  }
```

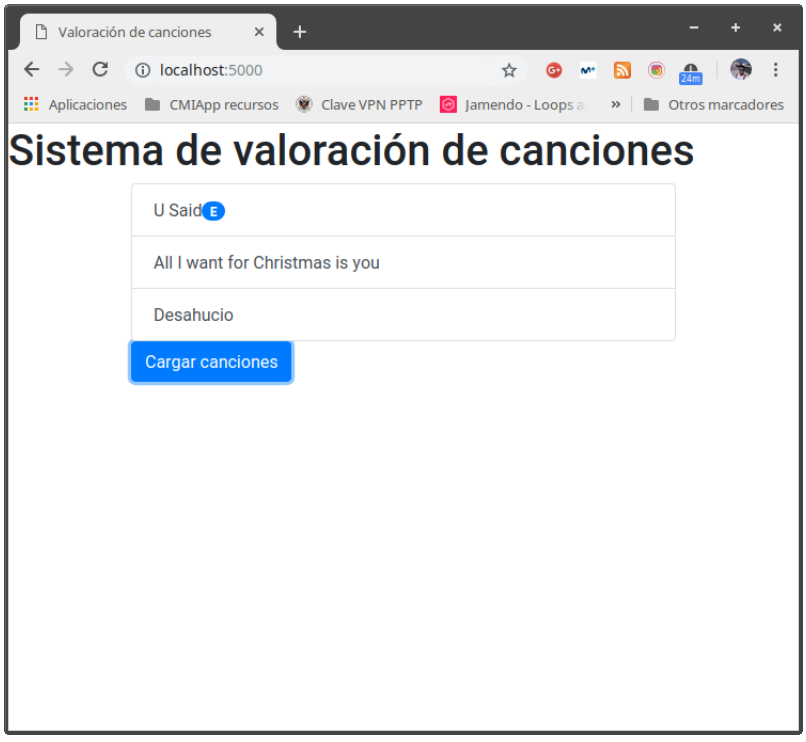
```
  }).then(res => res.json())
   .catch(error => console.error('Error:', error))
   .then(function(response) {
    console.log('Success:', response)
    if (response['estado']=='20001') {
     alert("No puedes valorar una canción sin escucharla");
    }
    if (response['estado']=='500') {
     alert("Ya has valorado esta canción");
    }
   });
}
```

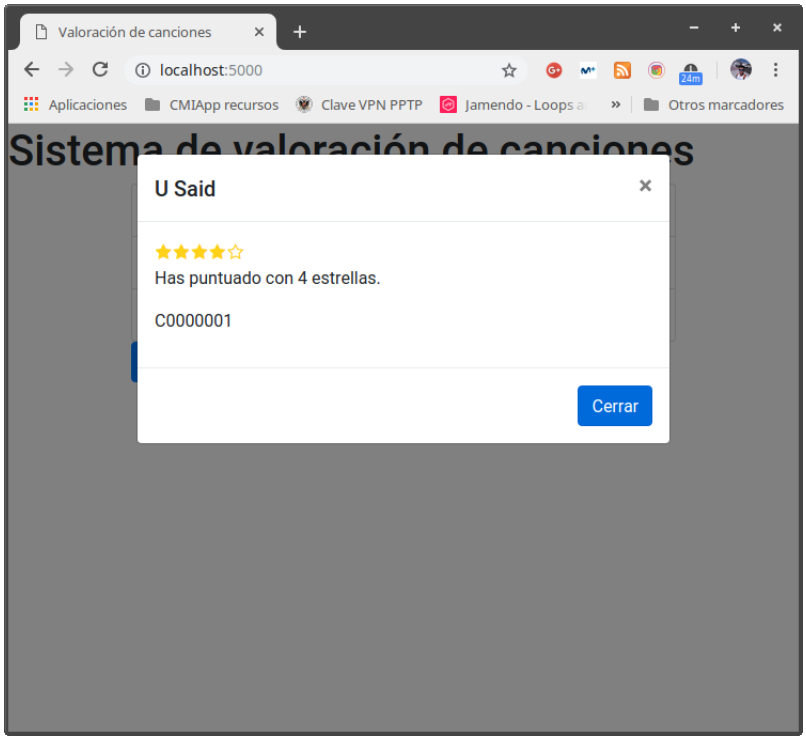
Para actualizar las valoraciones de cada canción después de que se haya añadido una valoración nueva se ha implementado un nuevo disparador que actualiza la valoración en la tabla canción después de cada INSERT en la tabla valora.

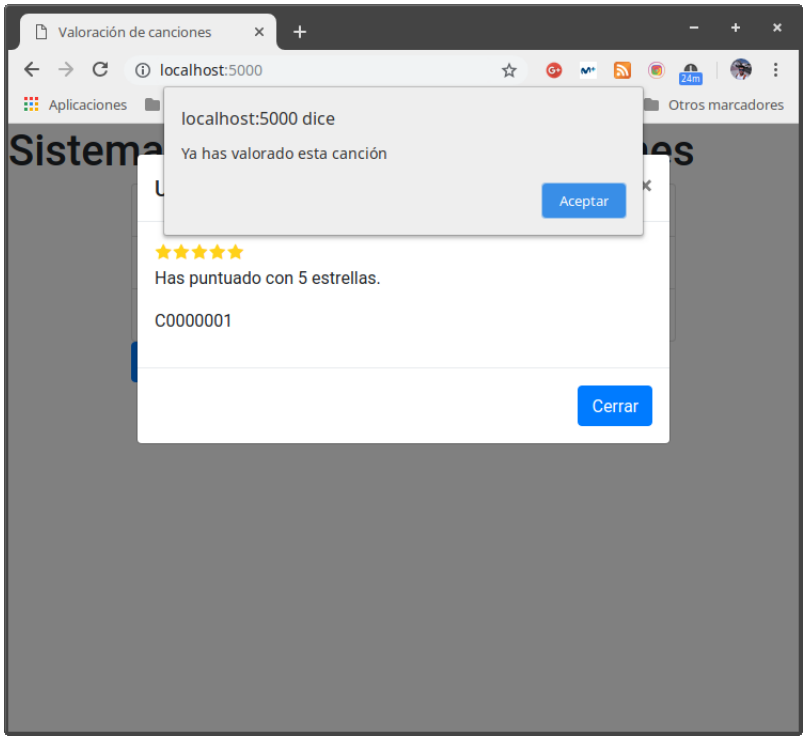
```
create or replace TRIGGER VALORA_TRIGG
AFTER INSERT on valora
DECLARE
 nueva_val NUMBER;
BEGIN
 FOR i IN (SELECT
 "A1"."ID_CANCION"        "ID_CANCION",
 "A1"."NOMBRE_CANCION"    "NOMBRE_CANCION",
 "A1"."GENERO"            "GENERO",
 "A1"."DURACION_SEG"      "DURACION_SEG",
 "A1"."FECHA_C"           "FECHA_C",
 "A1"."RUTA_AUDIO"        "RUTA_AUDIO",
 "A1"."NUM_REPRO"         "NUM_REPRO",
 "A1"."VALORACION"        "VALORACION"
FROM
 "X7151952"."CANCION" "A1") LOOP
  SELECT AVG(puntuacion) INTO nueva_val FROM valora
  WHERE id_cancion = i.id_cancion;
  IF nueva_val IS NULL THEN
   nueva_val := 0;
  END IF;
  UPDATE cancion SET valoracion = nueva_val
  WHERE id_cancion = i.id_cancion;
 END LOOP;

END;
```

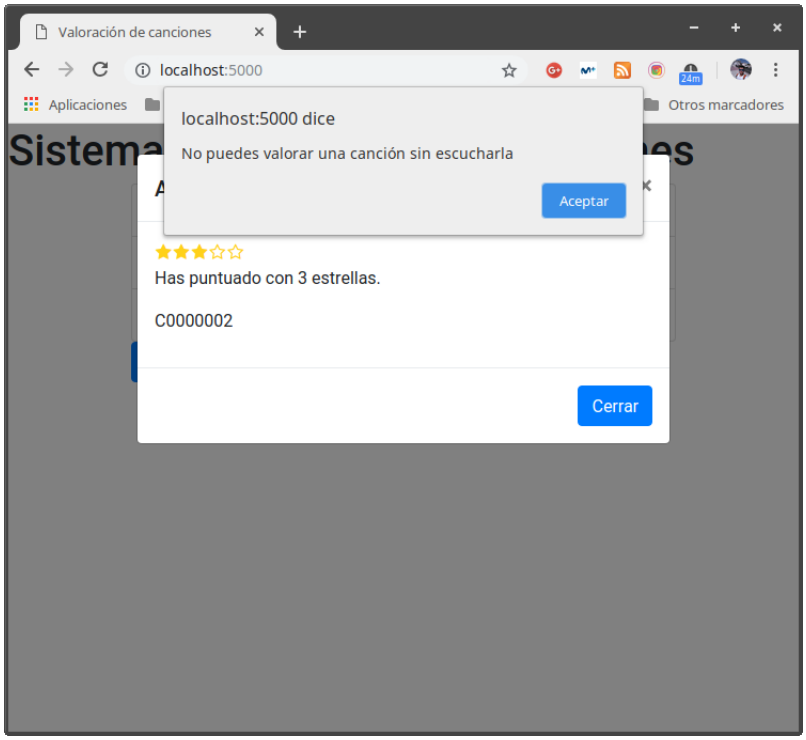
La aplicación tiene la siguiente apariencia:






Valoración de canciones
localhost:5000
Aplicaciones
CMIApp recursos
Clave VPN PPTP
Jamendo - Loops
Otros marcadores
Sistema de valoración de canciones
U Said
Has puntuado con 4 estrellas.
C0000001
Cerrar


Valoración de canciones
localhost:5000
Aplicaciones
Otros marcadores
localhost:5000 dice
Ya has valorado esta canción
Aceptar
Has puntuado con 5 estrellas.
C0000001
Cerrar



5. Conclusiones

Este es un trabajo realizado durante todo el cuatrimestre. Aunque todo lo esencial está contenido en este documento se pueden consultar los códigos generados en https://github.com/pedrodlz/DDSI_Practicas_1819