SERVICIO NACIONAL DE APRENDIZAJE SENA

Plan de trabajo para un aplicativo

Título: Desarrollo de aplicativo de gestión de álbumes musicales

Juliana Tique
David Solano
Jaider Moreno

09/06/2025

TABLA DE CONTENIDOS

```python
from app import db

class Usuario(db.Model):
    __tablename__ = 'usuarios'
    id_usuario = db.Column(db.Integer, primary_key=True)
    nombre = db.Column(db.String(100), nullable=False)
    contrasena = db.Column(db.String(100), nullable=False)

    # Relación con los álbumes
    albums = db.relationship('Album', backref='usuario', lazy=True)

    def __repr__(self):
        return f"<Usuario {self.nombre}>"
```



```python
from app import db

class Cancion(db.Model):
    __tablename__ = 'canciones'
    id_cancion = db.Column(db.Integer, primary_key=True)
    id_album = db.Column(db.Integer, db.ForeignKey('albums.id_album'), nullable=False)
    titulo_cancion = db.Column(db.String(200), nullable=False)
    duracion = db.Column(db.Integer, nullable=False)  # Duración en segundos
    interprete = db.Column(db.String(100), nullable=False)

    def __repr__(self):
        return f"<Cancion {self.titulo_cancion}>"
```

```python
from app import db

class Album(db.Model):
    __tablename__ = 'albums'
    id_album = db.Column(db.Integer, primary_key=True)
    id_usuario = db.Column(db.Integer, db.ForeignKey('usuarios.id_usuario'), nullable=False)
    titulo = db.Column(db.String(200), nullable=False)
    ano_produccion = db.Column(db.Integer, nullable=False)
    descripcion = db.Column(db.Text)
    medio = db.Column(db.String(50), nullable=False)

    # Relación con las canciones
    canciones = db.relationship('Cancion', backref='album', lazy=True)

    def __repr__(self):
        return f"<Album {self.titulo}>"
```



```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Bienvenido a la Colección Musical</title>
</head>
<body>
    <h1>Bienvenido a tu Colección Musical</h1>
    <a href="{{ url_for('login') }}">Iniciar sesión</a>
</body>
</html>
```

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Iniciar sesión</title>
</head>
<body>
    <h1>Iniciar sesión</h1>
    <form method="POST">
        <label for="nombre">Nombre:</label>
        <input type="text" id="nombre" name="nombre" required>
        <label for="contrasena">Contraseña:</label>
        <input type="password" id="contrasena" name="contrasena" required>
        <button type="submit">Iniciar sesión</button>
    </form>
</body>
</html>
```

```python
from flask import render_template, request, redirect, url_for
from app import app, db
from app.models.usuarios import Usuario

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        nombre = request.form['nombre']
        contrasena = request.form['contrasena']

        # Buscar el usuario
        usuario = Usuario.query.filter_by(nombre=nombre, contrasena=contrasena).first()

        if usuario:
            return redirect(url_for('mostrar_albumes', id_usuario=usuario.id_usuario))
        else:
            return 'Usuario o contraseña incorrectos'

    return render_template('login.html')
```

```python
from flask import render_template, request, redirect, url_for
from app import app, db
from app.models.albums import Album
from app.models.usuarios import Usuario

@app.route('/albums/<int:id_usuario>', methods=['GET'])
def mostrar_albumes(id_usuario):
    usuario = Usuario.query.get(id_usuario)
    albums = Album.query.filter_by(id_usuario=id_usuario).all()
    return render_template('albumes.html', usuario=usuario, albums=albums)

@app.route('/album/nuevo/<int:id_usuario>', methods=['GET', 'POST'])
def nuevo_album(id_usuario):
    if request.method == 'POST':
        titulo = request.form['titulo']
        ano_produccion = request.form['ano_produccion']
        descripcion = request.form['descripcion']
        medio = request.form['medio']

        nuevo_album = Album(id_usuario=id_usuario, titulo=titulo, ano_produccion=ano_produccion, 
        db.session.add(nuevo_album)
        db.session.commit()

        return redirect(url_for('mostrar_albumes', id_usuario=id_usuario))

    return render_template('nuevo_album.html', id_usuario=id_usuario)
```

```python
from flask import render_template, request, redirect, url_for
from app import app, db
from app.models.canciones import Cancion
from app.models.albums import Album

@app.route('/album/<int:id_album>/canciones', methods=['GET'])
def mostrar_canciones(id_album):
    album = Album.query.get(id_album)
    canciones = Cancion.query.filter_by(id_album=id_album).all()
    return render_template('canciones.html', album=album, canciones=canciones)

@app.route('/album/<int:id_album>/cancion/nueva', methods=['GET', 'POST'])
def nueva_cancion(id_album):
    if request.method == 'POST':
        titulo_cancion = request.form['titulo_cancion']
        duracion = request.form['duracion']  # Duración en segundos
        interprete = request.form['interprete']

        nueva_cancion = Cancion(id_album=id_album, titulo_cancion=titulo_cancion, duracion=duraci
        db.session.add(nueva_cancion)
        db.session.commit()

        return redirect(url_for('mostrar_canciones', id_album=id_album))

    return render_template('nueva_cancion.html', id_album=id_album)
```

Introducción

Lorenzo es un melómano y un coleccionista de música. Ha pasado toda su vida coleccionando álbumes musicales en discos, casetes y Cds, y tiene tantos, que tiene problemas para recordar en cuál álbum puede encontrar una canción que quiere escuchar.

Para solucionar este problema crea un sistema para almacenar la información de su colección y tener donde buscar en su colección de manera más fácil las canciones que quiere ubicar.

Su amigo vio su solución y le propuso convertirla en un servicio de almacenamiento de información musical en línea.

Con este propósito, Lorenzo ha decidido construir una aplicación web que le permita administrar sus álbumes y canciones. Este sistema debe ser capaz de permitir el registro de otros melómanos con un nombre y una contraseña. Cada usuario podrá tener un listado de álbumes para guardar por cada uno el título, año de producción, descripción, los artistas que interpretaron las canciones, las canciones que contiene y el medio sobre el cual está grabado.

El sistema le debe permitir a un usuario adicionar álbumes en cualquier momento. El sistema también le debe permitir modificar sus datos, e incluso borrar los álbumes si ingresa alguno repetido o que se dañó y ya no se puede escuchar.

Cuando un usuario elige un álbum de la lista, debe ver su información y las canciones que tiene. Para las canciones del álbum, Lorenzo necesita almacenar su título, duración en minutos y segundos y su intérprete. La idea que tiene es que en cualquier momento pueda editar la información o borrar las canciones, e incluso añadir canciones al registro

Valor agregado

Nosotros creemos que el valor agregador que le daremos al proyecto es la accesibilidad al ser un aplicativo web y la posibilidad de compartir mis listas de música

Metodología

Utilizaremos la metodología Scrum Agile para asegurar el buen funcionamiento del aplicativo con el cliente, para ello realizaremos tres reuniones con el cliente donde se hará el mantenimiento del proyecto referido a sus necesidades.

Fases

Planeación: en esta se llevará a cabo el levantamiento de información de lo que el usuario quiere ver dentro de su aplicativo y satisfacer las necesidades de cliente,

Requerimientos: Son los requerimientos que se obtuvieron del levantamiento de información y con los cuales se llevara acabo el funcionamiento de la pagina mediante estructuraciones que sean coherentes.

Diseño: En este diseñaran los requerimientos funcionales, mediante la implantación de estos en el estilo, colores y estructura

Implementación con código: teniendo en cuenta el estilo y estructura que se quiere llevar a cabo es punto de partida a la implementación del código para realizar los estilos y funcionalidades que el aplicativo quiere brindar

Pruebas y documentación: Mediante la creación del código es de suma importancia hacer pruebas de que sea intuitivo y amigable con el usuario y lo mas importante de todo que este 100% funcional y sin incoherencias

Despliegue: Al pasar las pruebas ya el aplicativo puede estar en funcionamiento dentro de la nube o plataformas digitales

Mantenimiento: Es necesario tener en cuanta que se necesita tener versiones y manejar ciertos arreglos ya que puede cosas mínimas que estén faltando o falte implementar como puede ser que no colapse la pagina con mas de 1.000 personas

Plan de trabajo concertado

Scrum: Jaider Moreno, Juliana Tique, David Solano

FECHA DE INICIO DEL SPRINT	DÍAS	PROGRESO	ESTE SPRINT
06/03/25	8	0%	25

TAREAS

CATEGORÍA	FUNCIÓN	ASIGNADA A	ACCIÓN	FUNDAMENTO	ESTADO	PRIORIDAD	PUNTOS	HORAS	INICIO	FINALIZACIÓN	DURACIÓN (DÍAS)	NOTAS Y COMENTARIOS
Investigación	Scrum master	Equipo	Levantamiento de información		Pruebas	Alto	2	0,50	3/06/2025	4/06/2025	1	Se realizará una entrevista con el cliente para saber sus requerimientos
Investigación	Desarrolladora	Jaider M.	Investigación Metodologías		En curso	Medio	4	1,00	3/06/2025	5/06/2025	2	Se realizará una daily para plantear las metodologías
Investigación		Equipo	Capacitación		Listo para empezar	Bajo	5	2,00	5/06/2025	6/06/2025	1	Se organizará una capacitación del personal en metodologías
Función	Product Owner	Jaider M.	Programación de calendario		Completado	Medio	6	0,25	6/06/2025	7/06/2025	1	Se creará una programación de calendario en días de los sprint
Contenido	Desarrolladora	David S.	Implementación		Pruebas	Medio	3	0,25	7/06/2025	8/06/2025	1	Se realizara una reunion con los clientes para saber si se esta cumpliendo con lo solicitado
Tarea	Desarrolladora	Juliana T.	Codificación		En curso	Alto	1	3,00	8/06/2025	9/06/2025	1	Se realizara la codificación del aplicativo Funcional
Función	QA	Juliana T.	Pruebas		Completado	Medio	1	1,00	9/06/2025	9/06/2025	0	Se realizaran pruebas respectivas para evitar errores y incoherencias
Tarea		Equipo	Integración		Completado	Medio	2	2,00	9/06/2025	10/06/2025	1	Se realizara la integracion de aplicativo en la nube y plataformas
Actualizar	Desarrolladora	Juliana T.	Mantenimiento		En espera	Alto	1	0,25	10/06/2025	10/06/2025	0	Se realizara mantenimientos de las pruebas en integración
											0	
											0	
											0	
											0	
											0	
											0	
											0	
											0	
											0	
											0	
							25	10,25			8	

Levantamiento de información

CATEGORÍA	FUNCIÓN	ASIGNADA A	RF	RNF	ESTADO	PRIORIDAD	PUNTOS	HORAS	INICIO	FINALIZACIÓN	DURACIÓN (Horas)	Tecnología
Investigación	Equipo	Equipo	Gestión de inventario (CRUD)		En curso	Alto	0	0,00	12:00:00 a. m.	12:00:00 a. m.	10	SQL
Investigación	Equipo	Equipo	Gestión de usuarios		En curso	Medio	0	0,00	12:00:00 a. m.	12:00:00 a. m.	4	HTML SQL y Javascript
Investigación	Equipo	Equipo		Accesibilidad	Pruebas	Bajo	0	0,00			0	Javascript
Investigación	Equipo	Equipo		Intuitiva	Pruebas	Medio	0	0,00			0	Css, bootstrap
Investigación	Equipo	Equipo		Seguridad	Pruebas	Medio	0	0,00			0	Http
Investigación	Equipo	Equipo	Gestión por Categorías		En curso	Alto	0	0,00			0	SQL
Investigación	Equipo	Equipo	Gestión de información		En espera	Medio	0	0,00			0	SQL
							0	0,00			14	

Investigación de metodología

CATEGORÍA	FUNCIÓN	ASIGNADA A	ACCIÓN	ESTADO	PRIORIDAD	PUNTOS	HORAS	INICIO	FINALIZACIÓN	DURACIÓN (DÍAS)	Vista
Gestión de inventario (CRUD)	Equipo	Jaider M.	Crear, modificar, eliminar y leer	En curso	Alto	0	0,00	3/09/2028	13/09/2028	10	Usuario.html
Gestión de usuarios	Equipo	Juliana T.	Administrar el acceso de usuarios	En curso	Medio	0	0,00	3/09/2028	7/09/2028	4	Administrador.html
Accesibilidad	Equipo	David S.	Facilitar el acceso a la cuenta del cliente	Pruebas	Bajo	0	0,00			0	Login.html
Intuitiva	Equipo	Jaider M.	Facilitar el funcionamiento de aplicativo	Pruebas	Bajo	0	0,00			0	N/A
Seguridad	Equipo	Juliana T.	Gestionar metodos seguros de acceso	Pruebas	Alto	0	0,00			0	N/A
Gestión por Categorías	Equipo	David S.	Filtrar por categorías las entradas	En curso	Medio	0	0,00			0	Catalogo.html
Gestión de información	Equipo	Jaider M.	Administrar la información de la entrada	En espera	Medio	0	0,00			0	Catalogo.html
						0	0,00			14	

Estructura de la base de datos

A continuación, es especifica la relación de los usuarios en la base de datos:

usuarios:

Almacena los usuarios del sistema, con su nombre y contraseña.

albums:

Almacena los álbumes de música, con su título, año de producción, descripción y el medio de almacenamiento (disco, casete, CD, etc.). Esta tabla también tiene una relación con los usuarios, ya que cada álbum pertenece a un usuario.

artistas:

Almacena los artistas. Esta tabla se usa para almacenar artistas de forma independiente.

album_artistas:

Relaciona los álbumes con los artistas. Un álbum puede tener varios artistas, y un artista puede participar en varios álbumes. Esta tabla tiene una relación de muchos a muchos.

canciones: Almacena las canciones de los álbumes, incluyendo el título de la canción, la duración (en segundos) y el intérprete.

usuario_album:

Relaciona los usuarios con sus álbumes. Un usuario puede tener varios álbumes y un álbum puede pertenecer a varios usuarios, lo cual se maneja en esta tabla de relación.

Funcionalidades del sistema:

Registrar un usuario: Al crear un usuario, se insertan los datos en la tabla usuarios.

Registrar un álbum:

El usuario puede crear un álbum y asociarlo a su cuenta en la tabla albums.

Agregar canciones al álbum:

Las canciones se asocian a un álbum mediante la tabla canciones.

Modificar álbumes:

Se pueden modificar los registros de los álbumes (como título, descripción, etc.).

Eliminar álbumes o canciones:

Los usuarios pueden eliminar álbumes o canciones y el sistema debe mantener la integridad referencial, eliminando registros relacionados si es necesario

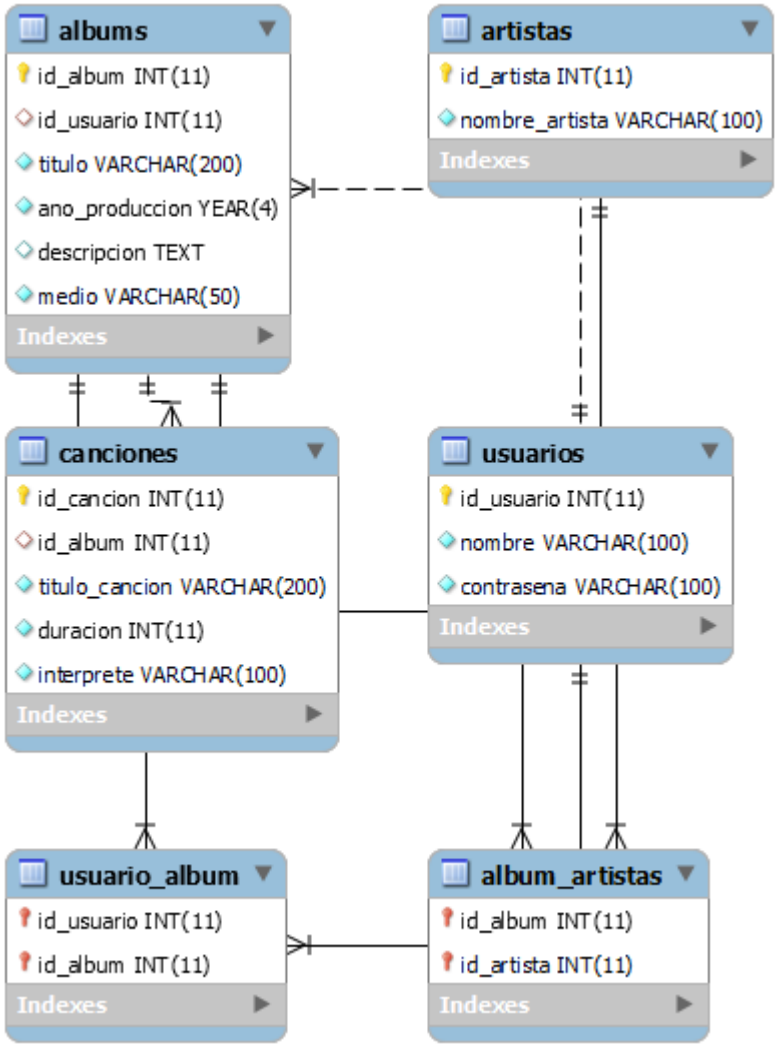

Figura 1. Entidad relación base de datos

Patrón de arquitectura

¿Qué es un patrón de arquitectura?

Según (Renato, 2023)

En términos generales, un patrón de arquitectura es la conceptualización de una solución genérica y reutilizable, aplicable a un problema de diseño de software en un contexto determinado, satisfaciendo las necesidades del negocio.

Los patrones arquitectónicos son la representación de las buenas prácticas y estructuras de diseño probadas, de modo que puedan reutilizarse. El objetivo es reutilizar las experiencias y conocimientos arquitectónicos que han dado buenos resultados en el pasado.

¿Por qué son importantes estos patrones?

Diversificación de tareas:

Al trabajar con estos patrones, se permite la fácil lectura del código, lo cual es útil en grupos de trabajo de gran tamaño y su cooperación.

Reutilización de código:

Permite que el software sea escalable y reutilizable, ahorra tiempo.

Mantenimiento y testing:

Facilita el mantenimiento del código ya que la lógica de presentación y de negocio están separadas.

Modelo vista controlador

Modelo-Vista-Controlador (MVC):

La arquitectura Model-View-Controller (MVC) es un patrón de diseño que separa una aplicación en tres componentes principales, que son:

- Modelo (Model). Este representa los datos y la lógica de negocio del sistema. Se encarga del acceso a los datos almacenados, ya sea en una base de datos o en otros medios. También define las reglas de negocio que se deben cumplir.

- Vista (View). Esta capa se encarga de la presentación de los datos. Por tanto, interactúa con el usuario, muestra la información y recibe la entrada. Se diseña para que sea independiente del modelo. Es decir, que, como en la hexagonal, se puede cambiar sin afectar a los datos subyacentes.

- Controlador (Controller). Es el intermediario entre los dos componentes anteriores. Procesa las solicitudes o entradas del usuario; las procesa y realiza las operaciones necesarias en el modelo, como por ejemplo, actualizarlo, y decide qué vista es la que se mostrará después con los resultados, (Arquitectura de Software: 5 Patrones Principales, 2024).

Este es uno de los patrones más conocidos. Separa la lógica de negocio (modelo), la interfaz de usuario (vista) y la lógica de control (controlador). Esto facilita la gestión de cambios y mejora la escalabilidad.

Desarrollo

```
/mi_app_musical
    /app
        __init__.py
        /models
            __init__.py
            usuarios.py
            albums.py
            canciones.py
        /views
            __init__.py
            home.html
            login.html
            albumes.html
            canciones.html
        /controllers
            __init__.py
            usuarios_controller.py
            albums_controller.py
            canciones_controller.py
        /config.py
    /venv
    /requirements.txt
    run.py
```

Figura 2. Estructura del proyecto

Modelos:

Los modelos representarán las tablas de la base de datos serán programados en Python de las tablas:

Usuarios:

```python
from app import db

class Usuario(db.Model):
    __tablename__ = 'usuarios'
    id_usuario = db.Column(db.Integer, primary_key=True)
    nombre = db.Column(db.String(100), nullable=False)
    contrasena = db.Column(db.String(100), nullable=False)

    # Relación con los álbumes
    albums = db.relationship('Album', backref='usuario', lazy=True)

    def __repr__(self):
        return f"<Usuario {self.nombre}>"
```

Canciones

```python
from app import db

class Cancion(db.Model):
    __tablename__ = 'canciones'
    id_cancion = db.Column(db.Integer, primary_key=True)
    id_album = db.Column(db.Integer, db.ForeignKey('albums.id_album'), nullable=False)
    titulo_cancion = db.Column(db.String(200), nullable=False)
    duracion = db.Column(db.Integer, nullable=False)  # Duración en segundos
    interprete = db.Column(db.String(100), nullable=False)

    def __repr__(self):
        return f"<Cancion {self.titulo_cancion}>"
```

Albums

```python
from app import db

class Album(db.Model):
    __tablename__ = 'albums'
    id_album = db.Column(db.Integer, primary_key=True)
    id_usuario = db.Column(db.Integer, db.ForeignKey('usuarios.id_usuario'), nullable=False)
    titulo = db.Column(db.String(200), nullable=False)
    ano_produccion = db.Column(db.Integer, nullable=False)
    descripcion = db.Column(db.Text)
    medio = db.Column(db.String(50), nullable=False)

    # Relación con las canciones
    canciones = db.relationship('Cancion', backref='album', lazy=True)

    def __repr__(self):
        return f"<Album {self.titulo}>"
```

Vista:

Casa

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Bienvenido a la Colección Musical</title>
</head>
<body>
    <h1>Bienvenido a tu Colección Musical</h1>
    <a href="{{ url_for('login') }}">Iniciar sesión</a>
</body>
</html>
```

login

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Iniciar sesión</title>
</head>
<body>
    <h1>Iniciar sesión</h1>
    <form method="POST">
        <label for="nombre">Nombre:</label>
        <input type="text" id="nombre" name="nombre" required>
        <label for="contrasena">Contraseña:</label>
        <input type="password" id="contrasena" name="contrasena" required>
        <button type="submit">Iniciar sesión</button>
    </form>
</body>
</html>
```

Controlador:

Usuarios

```python
from flask import render_template, request, redirect, url_for
from app import app, db
from app.models.usuarios import Usuario

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        nombre = request.form['nombre']
        contrasena = request.form['contrasena']

        # Buscar el usuario
        usuario = Usuario.query.filter_by(nombre=nombre, contrasena=contrasena).first()

        if usuario:
            return redirect(url_for('mostrar_albumes', id_usuario=usuario.id_usuario))
        else:
            return 'Usuario o contraseña incorrectos'

    return render_template('login.html')
```

Albums

```python
from flask import render_template, request, redirect, url_for
from app import app, db
from app.models.albums import Album
from app.models.usuarios import Usuario

@app.route('/albums/<int:id_usuario>', methods=['GET'])
def mostrar_albumes(id_usuario):
    usuario = Usuario.query.get(id_usuario)
    albums = Album.query.filter_by(id_usuario=id_usuario).all()
    return render_template('albumes.html', usuario=usuario, albums=albums)

@app.route('/album/nuevo/<int:id_usuario>', methods=['GET', 'POST'])
def nuevo_album(id_usuario):
    if request.method == 'POST':
        titulo = request.form['titulo']
        ano_produccion = request.form['ano_produccion']
        descripcion = request.form['descripcion']
        medio = request.form['medio']

        nuevo_album = Album(id_usuario=id_usuario, titulo=titulo, ano_produccion=ano_produccion, 
        db.session.add(nuevo_album)
        db.session.commit()

        return redirect(url_for('mostrar_albumes', id_usuario=id_usuario))

    return render_template('nuevo_album.html', id_usuario=id_usuario)
```

Canciones

```python
from flask import render_template, request, redirect, url_for
from app import app, db
from app.models.canciones import Cancion
from app.models.albums import Album

@app.route('/album/<int:id_album>/canciones', methods=['GET'])
def mostrar_canciones(id_album):
    album = Album.query.get(id_album)
    canciones = Cancion.query.filter_by(id_album=id_album).all()
    return render_template('canciones.html', album=album, canciones=canciones)

@app.route('/album/<int:id_album>/cancion/nueva', methods=['GET', 'POST'])
def nueva_cancion(id_album):
    if request.method == 'POST':
        titulo_cancion = request.form['titulo_cancion']
        duracion = request.form['duracion']  # Duración en segundos
        interprete = request.form['interprete']

        nueva_cancion = Cancion(id_album=id_album, titulo_cancion=titulo_cancion, duracion=duraci
        db.session.add(nueva_cancion)
        db.session.commit()

        return redirect(url_for('mostrar_canciones', id_album=id_album))

    return render_template('nueva_cancion.html', id_album=id_album)
```

Flujo de datos en MVC

1. El usuario interactúa con la vista.
2. El controlador recibe la entrada del usuario.
3. El controlador puede actualizar el modelo con la información recibida o solicitar datos al modelo.
4. El modelo proporciona los datos actualizados al controlador.
5. El controlador decide qué vista debe mostrar y le envía los datos adecuados.
6. La vista se actualiza con los datos proporcionados por el controlador.
7. El usuario ve los cambios en la interfaz de usuario.

(Que Es El Flujo Mcv En Desarrollo de Software - Buscar Con Google, 2024)

Conclusiones

Bootstrap permite un cómodo desarrollo del front-end por medio de clases ya programadas, entre ellas las grillas, que nos permite manejar la distribución de la página, para hacerla mucho más amigable con la experiencia del usuario.

El uso de tamaños automáticos facilita el desarrollo de la aplicación, pero dificulta el funcionamiento del aplicativo a todas las plataformas, por ejemplo, al momento de manejar imágenes.

Al momento de usar el framework Bootstrap es útil trabajarlo por medio del link de referencia más que descargarlo manualmente.

Bibliografía

ChatGPT. (s/f). Chatgpt.com. Recuperado el 12 de mayo de 2025, de https://chatgpt.com/

Otto, M., & Thornton, J. (s/f-a). *Columns*. Getbootstrap.com. Recuperado el 12 de mayo de 2025, de https://getbootstrap.com/docs/5.3/layout/columns/

Otto, M., & Thornton, J. (s/f-b). *Grid system*. Getbootstrap.com. Recuperado el 12 de mayo de 2025, de https://getbootstrap.com/docs/4.0/layout/grid/