FACULTAD DE INGENIERÍA CARRERA DE INGENIERÍA INFORMÁTICA FACULTAD COMUNITARIA DE CAACUPÉ

MATERIA

Optativa

ALUMNO

Juan José Romero Mayeregger

DOCENTE

Ing. Ricardo Maidana

CURSO

5° año – 9no. semestre

Caacupé – Paraguay

2024

INTRODUCCION

En la actualidad, la creación de aplicaciones multimedia que proporcionen experiencias interactivas y personalizadas para los usuarios se ha vuelto esencial. Uno de los pilares en el desarrollo de aplicaciones de escritorio es la capacidad de gestionar interfaces gráficas de usuario (GUI) eficientes y atractivas. En este contexto, la biblioteca PyQt se destaca como una herramienta poderosa para el desarrollo de aplicaciones GUI en Python. PyQt es un conjunto de enlaces de Python para las bibliotecas Qt, que permiten la creación de aplicaciones gráficas avanzadas con una facilidad y flexibilidad notables.

El presente trabajo de investigación tiene como objetivo principal el desarrollo de un reproductor de música utilizando la biblioteca PyQt. Este proyecto no solo busca explorar las capacidades técnicas y funcionales de PyQt en la creación de interfaces de usuario, sino también integrar funcionalidades multimedia que permitan una experiencia de usuario rica y satisfactoria. La elección de PyQt se debe a su extensa documentación, su capacidad para soportar múltiples plataformas y su integración con Python, un lenguaje ampliamente utilizado por su simplicidad y potencia.

PyQT

PyQt es un conjunto de enlaces para Python que permite usar el conjunto de herramientas de interfaz gráfica de usuario (GUI) Qt de C++. PyQt es una de las bibliotecas más populares para crear aplicaciones GUI en Python debido a su potencia, flexibilidad y la gran cantidad de widgets y herramientas que proporciona.

Características

- Widgets y Componentes: Ofrece una amplia gama de widgets (botones, cuadros de texto, tablas, etc.) y componentes para construir interfaces de usuario complejas.
- Compatibilidad Multiplataforma: Las aplicaciones creadas con PyQt pueden ejecutarse en diferentes sistemas operativos como Windows, macOS y Linux sin cambiar el código fuente.
- Herramientas de Diseño: Incluye Qt Designer, una herramienta WYSIWYG (What You See Is What You Get) para diseñar interfaces gráficas visualmente.
- Flexibilidad: Permite usar tanto programación orientada a objetos como programación basada en señales y ranuras, que es un poderoso mecanismo de gestión de eventos en Qt.
- Documentación y Comunidad: Posee una extensa documentación y una activa comunidad que proporciona soporte y recursos.

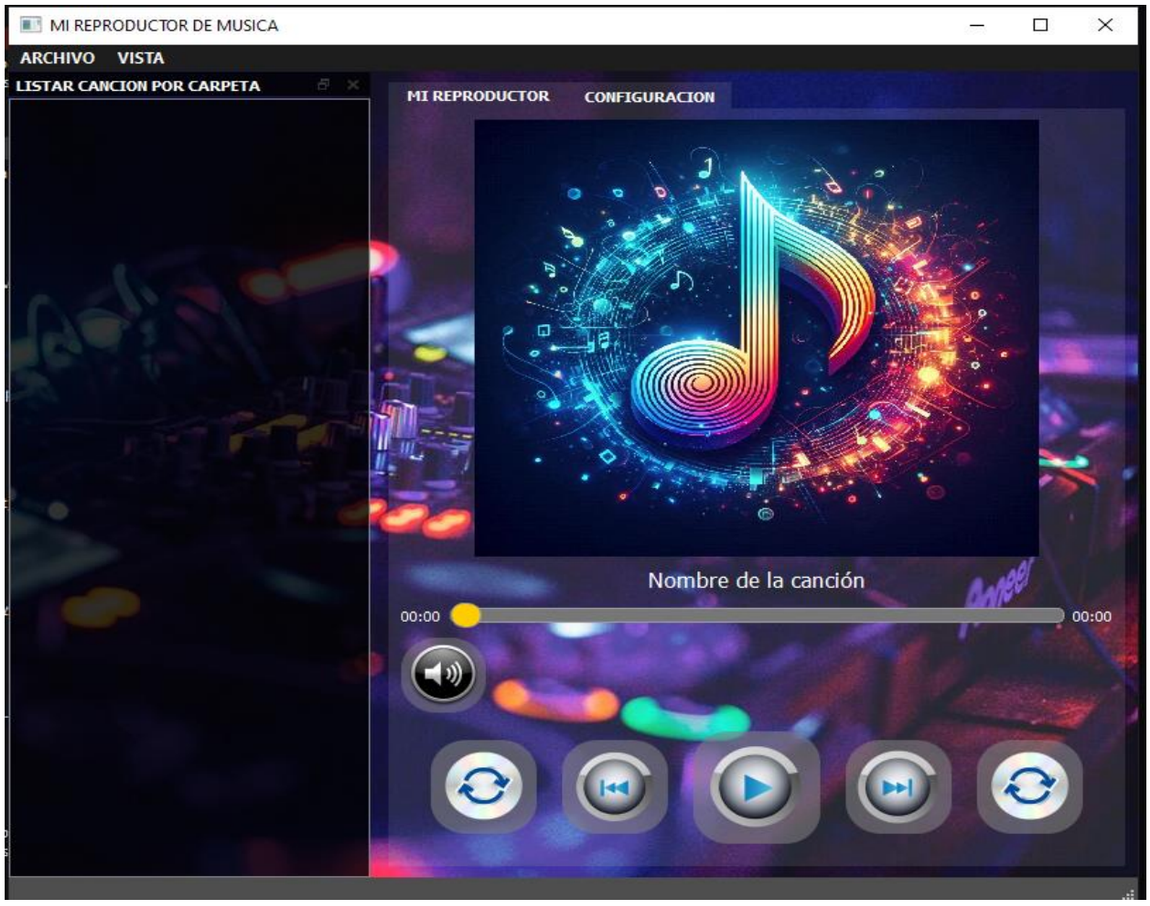
Por qué usar PyQT

Elegí la biblioteca PyQt para desarrollar aplicaciones con interfaz gráfica en Python debido a sus numerosas ventajas y capacidades. PyQt destaca por su potencia y flexibilidad, permitiendo la creación de aplicaciones complejas y altamente personalizables. Además, su compatibilidad multiplataforma asegura que las aplicaciones funcionen de manera consistente en diferentes sistemas operativos. Otra ventaja significativa es la inclusión de Qt Designer, una herramienta que facilita el diseño de interfaces gráficas de usuario de manera visual e intuitiva.

Reproductor de música

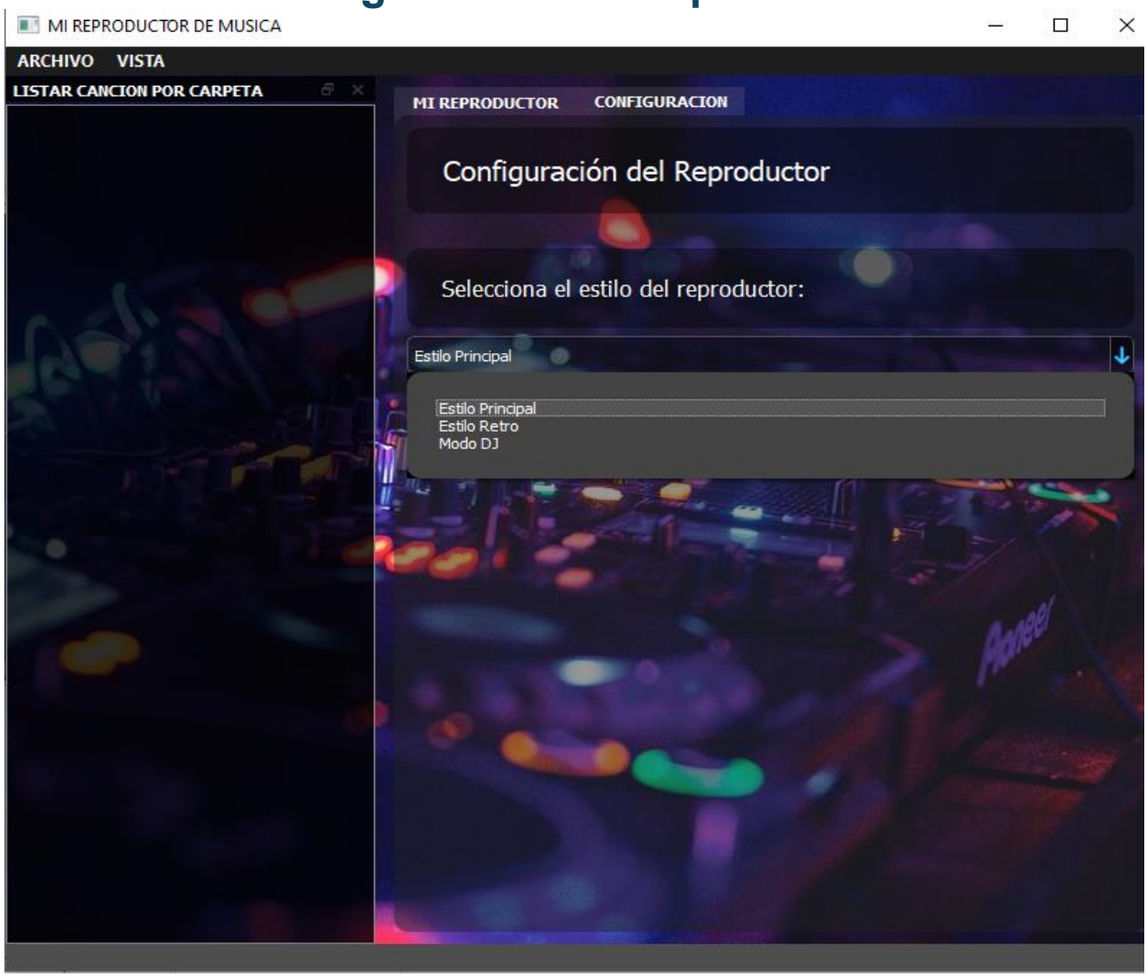
Desarrollaré un reproductor de música utilizando la biblioteca PyQt, empleando la herramienta de diseño Qt Designer y Python como lenguaje de programación. El objetivo del reproductor es ofrecer una experiencia de reproducción de música optimizada, permitiendo a los usuarios disfrutar de sus canciones favoritas a través de una interfaz gráfica amigable e intuitiva.

Interfaz Principal



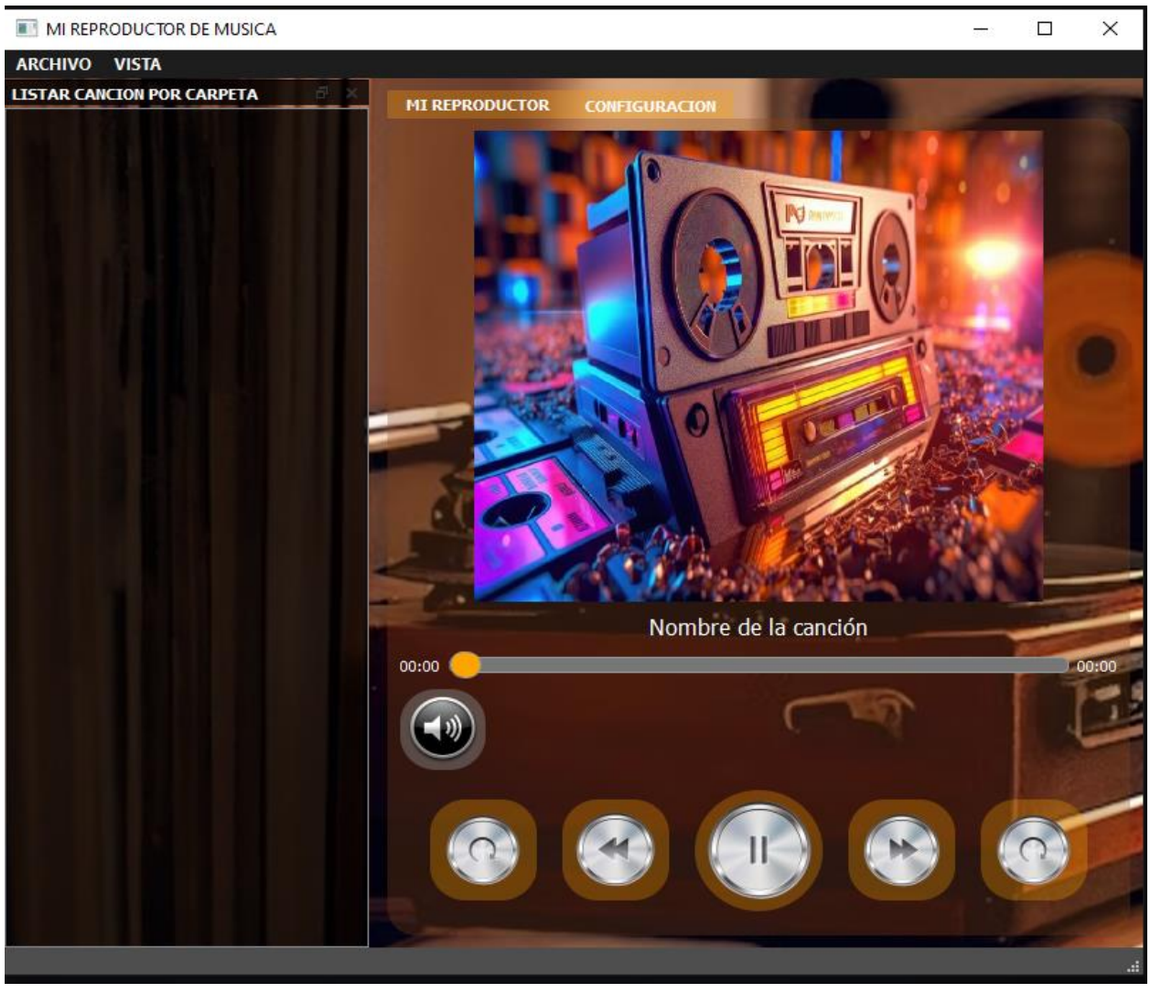
La interfaz del reproductor de música está diseñada para ser intuitiva y accesible, presentando controles de reproducción esenciales y un gestor de música integrado. Los controles de reproducción incluyen botones para reproducir, pausar, avanzar y retroceder, repetir y aleatorio, asegurando una interacción fluida y directa. Además, el gestor de música permite a los usuarios navegar y seleccionar fácilmente las pistas de su biblioteca, optimizando la experiencia auditiva y facilitando la gestión de listas de reproducción. Esta configuración técnica garantiza una experiencia de usuario enriquecida y eficiente, maximizando la funcionalidad del reproductor dentro de una interfaz simplificada.

Configuración del Reproductor

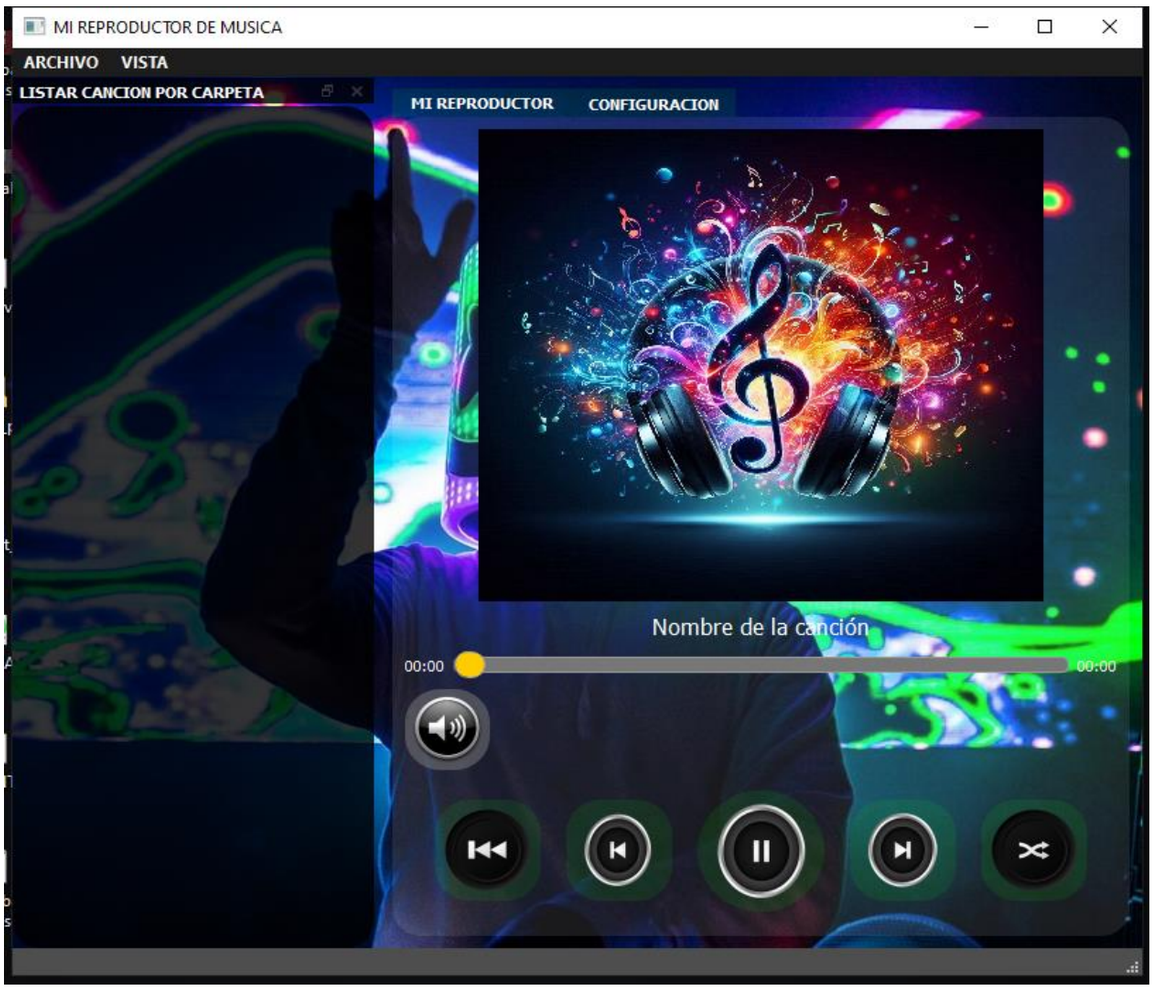


Además, el reproductor ofrece una ventana de configuración en donde el usuario puede seleccionar el estilo del reproductor que el desee, ofreciendo estilos como Estilo principal, Estilo retro y el modo dj, adaptándose a las necesidades del usuario.

Estilo Retro



Modo DJ



Código del reproductor

Importación de librerías necesarias.

```
# Importación de librerías necesarias
import sys
import os
import random
from PyQt5.QtWidgets import (QApplication, QMainWindow, QPushButton, QFileDialog,
QLabel,
                             QVBoxLayout, QWidget, QListWidget, QHBoxLayout,
QDockWidget,
                             QStatusBar, QTabWidget, QListWidgetItem, QAction,
QSlider,QComboBox)
from PyQt5.QtMultimedia import QMediaPlayer, QMediaContent
from PyQt5.QtGui import QPixmap, QKeySequence, QIcon, QDragEnterEvent, QDropEvent
from PyQt5.QtCore import Qt, QStandardPaths, QUrl, QSize
```

Inicializa la ventana principal del reproductor de música.

```
class MusicPlayer(QMainWindow):
    def __init__(self):
        """
        Inicializa la ventana principal del reproductor de música.
        """
        super().__init__()
        self.initialize_ui()
        self.status_Bar = QStatusBar()  # Crear barra de estado
        self.setStatusBar(self.status_Bar)  # Establecer la barra de estado
        self.current_music_folder = ""  # Inicializar carpeta de música
        current_dir = os.path.dirname(os.path.abspath(__file__))  # Obtiene el
directorio actual del archivo Python en ejecución
        img_path = os.path.join(current_dir, 'img', 'caratula_m.jpeg')  #
Construye la ruta completa para la imagen 'caratula_m.jpeg' en el subdirectorio
'img'
        pixmap = QPixmap(img_path)  # Carga la imagen desde la ruta especificada
en un objeto QPixmap
        if pixmap.isNull():  # Verifica si el QPixmap no pudo cargar la imagen
(es nulo)
            print(f"Could not create pixmap from {img_path}")  # Imprime un
mensaje de error si la imagen no pudo ser cargada

        css_path = os.path.join(current_dir, 'estilos.css')  # Construye la ruta
completa para el archivo de estilos CSS 'estilos.css'
        with open(css_path, 'r') as file:
            Style = file.read()  # Leer el archivo de estilos CSS
        self.setStyleSheet(Style)  # Aplicar los estilos CSS
```

```
        self.player = QMediaPlayer()  # Inicializar el reproductor multimedia
self.player.positionChanged.connect(self.actualizar_posicion)  # Conectar la
señal de cambio de posición
        self.player.durationChanged.connect(self.actualizar_duracion)  # Conectar
la señal de cambio de duración
        self.player.mediaStatusChanged.connect(self.handle_media_status_changed)
# Conecta la señal `mediaStatusChanged` del reproductor de medios a la función
`handle_media_status_changed`
        self.is_paused = False  # Estado de pausa
        self.current_song_index = -1  # Índice de la canción actual
```

Inicializa la interfaz de usuario del reproductor de música.

```
def initialize_ui(self):
        """
        Inicializa la interfaz de usuario del reproductor de música.
        """
        self.setGeometry(100, 100, 800, 350)  # Establecer geometría de la
ventana
        self.setWindowTitle("MI REPRODUCTOR DE MUSICA")  # Establecer el título
de la ventana
        self.generar_ventana()  # Generar la ventana principal
        self.crear_dock()  # Crear el dock de la lista de canciones
        self.crear_dock_arrastrar()
        self.crear_accion()  # Crear acciones
        self.crear_menu()  # Crear menú
        self.show()  # Mostrar la ventana
        screen_geo = QApplication.desktop().screenGeometry()  # Obtener geometría
de la pantalla
        x = (screen_geo.width() - self.width()) // 2  # Calcular posición X para
centrar la ventana
        y = (screen_geo.height() - self.height()) // 2  # Calcular posición Y
para centrar la ventana
        self.move(x, y)  # Mover la ventana a la posición calculada
        self.show()  # Mostrar la ventana
```

Genera las pestañas y el contenedor principal de la ventana.

```
def generar_ventana(self):
        """
        Genera las pestañas y el contenedor principal de la ventana.
        """
        tab_bar = QTabWidget(self)  # Crear barra de pestañas
        self.reproductor_container = QWidget()  # Crear contenedor para la
pestaña del reproductor
```

```
        self.configuracion_container = QWidget()  # Crear contenedor para la
pestaña de configuración tab_bar.addTab(self.reproductor_container, "  MI
REPRODUCTOR  ")  # Añadir pestaña del reproductor
        tab_bar.addTab(self.configuracion_container, " CONFIGURACION ")  # Añadir
pestaña de configuración (comentado)

        self.generar_reproductor_tab()  # Generar el contenido de la pestaña del
reproductor
        self.generar_configuracion_tab()

        tab_h_box = QHBoxLayout()  # Crear layout horizontal para las pestañas
        tab_h_box.addWidget(tab_bar)  # Añadir la barra de pestañas al layout

        main_container = QWidget()  # Crear contenedor principal
        main_container.setLayout(tab_h_box)  # Establecer el layout para el
contenedor principal
        self.setCentralWidget(main_container)  # Establecer el contenedor
principal como el widget central
```

Genera la pestaña del reproductor de música.

```
def generar_reproductor_tab(self):
        """
        Genera la pestaña del reproductor de música.
        """
        main_v_box = QVBoxLayout()  # Crear layout vertical principal
        boton_h_box = QHBoxLayout()  # Crear layout horizontal para los botones

        self.fondo_imagen = QLabel()  # Crear etiqueta para la caratula de la
musica
        pixmap = QPixmap("img/caratula_m.jpeg").scaled(400, 350)  # Cargar y
escalar la caratula de la musica
        self.fondo_imagen.setPixmap(pixmap)  # Establecer la caratula de la
musica
        self.fondo_imagen.setScaledContents(False)  # No escalar el contenido
        self.fondo_imagen.setAlignment(Qt.AlignCenter)  # Alinear la caratula al
centro
        self.boton_repetir = QPushButton("")  # Crear botón de repetir
        self.boton_repetir.setObjectName('boton_repetir')  # Establecer el nombre
del objeto
        self.boton_retroceder = QPushButton("")  # Crear botón de retroceder
        self.boton_retroceder.setObjectName('boton_retroceder')  # Establecer el
nombre del objeto
        self.boton_play = QPushButton("")  # Crear botón de play/pausa
```

```
        self.boton_play.setObjectName('boton_play')  # Establecer el nombre del objeto
        self.boton_avanzar = QPushButton("")  # Crear botón de avanzar
        self.boton_avanzar.setObjectName('boton_avanzar')  # Establecer el nombre del objeto
        self.boton_aleatorio = QPushButton("")  # Crear botón de aleatorio
        self.boton_aleatorio.setObjectName('boton_aleatorio')  # Establecer el nombre del objeto

        # Establecer el tamaño de los botones
        self.boton_repetir.setFixedSize(75, 75)
        self.boton_retroceder.setFixedSize(75, 75)
        self.boton_play.setFixedSize(90, 90)
        self.boton_avanzar.setFixedSize(75, 75)
        self.boton_aleatorio.setFixedSize(75, 75)


        self.song_name_label = QLabel("Nombre de la canción")  # Crear etiqueta para el nombre de la canción
        self.song_name_label.setAlignment(Qt.AlignCenter)  # Alinear el texto al centro
        self.song_name_label.setStyleSheet("color: white; font-size: 16px; ")  # Establecer el estilo de la etiqueta

        self.progress_bar = QSlider(Qt.Horizontal)  # Crear barra de progreso horizontal
        self.progress_bar.setRange(0, 100)  # Establecer el rango de la barra de progreso
        self.progress_bar.sliderMoved.connect(self.set_position)  # Conectar el movimiento del slider a la función set_position
        self.progress_bar.sliderPressed.connect(self.slider_pressed)  # Conectar la presión del slider a la función slider_pressed
        self.progress_bar.sliderReleased.connect(self.slider_released)  # Conectar la liberación del slider a la función slider_released

        self.volume_slider = QSlider(Qt.Horizontal)  # Crear slider de volumen horizontal
        self.volume_slider.setRange(0, 100)  # Establecer el rango del slider de volumen
        self.volume_slider.setValue(100)  # Establecer el valor inicial del volumen al máximo
        self.volume_slider.valueChanged.connect(self.set_volume)  # Conectar el cambio de valor del slider a la función set_volume
        self.volume_slider.hide()  # Ocultar inicialmente la barra de volumen
```

```
        self.current_time_label = QLabel("00:00")  # Crear etiqueta para el tiempo actual de reproducción
        self.current_time_label.setStyleSheet("color: white;")  # Establecer el estilo de la etiqueta
        self.total_time_label = QLabel("00:00")  # Crear etiqueta para el tiempo total de reproducción
        self.total_time_label.setStyleSheet("color: white;")  # Establecer el estilo de la etiqueta

        progress_layout = QHBoxLayout()  # Crear layout horizontal para la barra de progreso
        progress_layout.addWidget(self.current_time_label)  # Añadir la etiqueta del tiempo actual al layout
        progress_layout.addWidget(self.progress_bar)  # Añadir la barra de progreso al layout
        progress_layout.addWidget(self.total_time_label)  # Añadir la etiqueta del tiempo total al layout

        # Icono de volumen con estilo
        self.icono_volumen = QPushButton()  # Crear botón para el icono de volumen
        self.icono_volumen.setIcon(QIcon("img/volumen_icono.png"))  # Establecer el icono del botón
        self.icono_volumen.setIconSize(QSize(50, 50))  # Establecer el tamaño del icono
        self.icono_volumen.setFixedSize(60, 60)  # Establecer el tamaño del botón
        self.icono_volumen.setStyleSheet("""
            QPushButton {
                background-color: rgba(128, 128, 128, 0.5);
                border: none;
                border-radius: 25px;
                padding: 10px;
            }
        """)  # Establecer el estilo del botón
        self.icono_volumen.clicked.connect(self.toggle_volume_slider)  # Conectar el clic del botón a la función toggle_volume_slider

        volume_layout = QHBoxLayout()  # Crear layout horizontal para el control de volumen
        volume_layout.addWidget(self.icono_volumen)  # Añadir el botón del icono de volumen al layout
        volume_layout.addWidget(self.volume_slider)  # Añadir el slider de volumen al layout
        volume_layout.setAlignment(Qt.AlignLeft)  # Alinear el layout a la izquierda
```

```
        boton_h_box.addWidget(self.boton_repetir)  # Añadir botón de repetir al layout de botones
        boton_h_box.addWidget(self.boton_retroceder)  # Añadir botón de retroceder al layout de botones
        boton_h_box.addWidget(self.boton_play)  # Añadir botón de play/pausa al layout de botones
        boton_h_box.addWidget(self.boton_avanzar)  # Añadir botón de avanzar al layout de botones
        boton_h_box.addWidget(self.boton_aleatorio)  # Añadir botón de aleatorio al layout de botones
        boton_container = QWidget()  # Crear contenedor para los botones
        boton_container.setLayout(boton_h_box)  # Establecer el layout para el contenedor de botones


        main_v_box.addWidget(self.fondo_imagen)  # Añadir la imagen de fondo al layout principal
        main_v_box.addWidget(self.song_name_label)  # Añadir la etiqueta del nombre de la canción al layout principal
        main_v_box.addLayout(progress_layout)  # Añadir el layout de la barra de progreso al layout principal
        main_v_box.addLayout(volume_layout)  # Añadir el layout del control de volumen al layout principal
        main_v_box.addWidget(boton_container)  # Añadir el contenedor de botones al layout principal

        self.reproductor_container.setLayout(main_v_box)  # Establecer el layout principal en el contenedor del reproductor
        # Conectar botones a funciones
        self.boton_repetir.clicked.connect(self.repetir)  # Conectar botón de repetir a la función repetir
        self.boton_retroceder.clicked.connect(self.retroceder)  # Conectar botón de retroceder a la función retroceder
        self.boton_play.clicked.connect(self.play_pause)  # Conectar botón de play/pausa a la función play_pause
        self.boton_avanzar.clicked.connect(self.avanzar)  # Conectar botón de avanzar a la función avanzar
        self.boton_aleatorio.clicked.connect(self.aleatorio)  # Conectar botón de aleatorio a la función aleatorio
```

Maneja los cambios de estado de los medios y detecta errores.

```
def handle_media_status_changed(self, status):
        """
        Maneja los cambios de estado de los medios y detecta errores.
        """
        if status == QMediaPlayer.InvalidMedia:  # Comprueba si el estado del
medio es inválido
            error_message = "ERROR! La música no puede ser reproducida"  # Define
el mensaje de error para un medio inválido
            self.song_name_label.setText(error_message)  # Actualiza el texto de
la etiqueta de nombre de la canción con el mensaje de error
            self.song_name_label.setStyleSheet("color: red; font-size: 16px;
background-color: black;")  # Cambia el estilo de la etiqueta para reflejar el
error
        elif status == QMediaPlayer.NoMedia:  # Comprueba si no hay ningún medio
cargado
            error_message = "No se ha cargado ningún archivo de música"  # Define
el mensaje de error para cuando no hay ningún medio cargado
            self.song_name_label.setText(error_message)  # Actualiza el texto de
la etiqueta de nombre de la canción con el mensaje de error
            self.song_name_label.setStyleSheet("color: red; font-size: 16px;
background-color: #5E5E5E;")  # Cambia el estilo de la etiqueta para reflejar que
no hay ningún medio
        else:
            self.song_name_label.setStyleSheet("color: white; font-size:
16px;")  # Restaura el estilo de la etiqueta para los otros estados
```

Alterna la visibilidad del slider de volumen.

```
def toggle_volume_slider(self):
        """
        Alterna la visibilidad del slider de volumen.
        """
        if self.volume_slider.isVisible():  # Comprobar si el slider de volumen
es visible
            self.volume_slider.hide()  # Ocultar el slider de volumen
        else:
            self.volume_slider.show()  # Mostrar el slider de volumen
```

Crea las acciones del menú de la aplicación.

```
def crear_accion(self):
        """
        Crea las acciones del menú de la aplicación.
        """

        self.listar_musica_accion = QAction('LISTAR MUSICA', self,
checkable=True)  # Crear acción para listar la música
        self.listar_musica_accion.setShortcut(QKeySequence("Ctrl+L"))  #
Establecer atajo de teclado para la acción
        self.listar_musica_accion.setStatusTip("AQUI PUEDES LISTAR O NO LA MUSICA
A REPRODUCIR")  # Establecer texto de estado
        self.listar_musica_accion.triggered.connect(self.listar_musica)  #
Conectar la acción a la función listar_musica
        self.listar_musica_accion.setChecked(True)  # Establecer la acción como
seleccionada inicialmente

        self.abrir_archivo_musica_action = QAction('ABRIR CARPETA', self)  #
Crear acción para abrir carpeta de música
        self.abrir_archivo_musica_action.setShortcut(QKeySequence("Ctrl+O"))  #
Establecer atajo de teclado para la acción
        self.abrir_archivo_musica_action.setStatusTip("AGREGE TU CARPETA DE
MUSICA")  # Establecer texto de estado
        self.abrir_archivo_musica_action.triggered.connect(self.abrir_carpeta_mus
ica)  # Conectar la acción a la función abrir_carpeta_musica

        self.arrastrar_musica_accion = QAction("ARRASTRAR MUSICA", self,
checkable=True ) # Crear acción para abrir carpeta de música
        self.arrastrar_musica_accion.setShortcut(QKeySequence("Ctrl+A")) #
Establecer atajo de teclado para la acción
        self.arrastrar_musica_accion.setStatusTip("AQUI PUEDES ARRASTAR Y SOLTAR
LA MUSICA") # Establecer texto de estado
        self.arrastrar_musica_accion.triggered.connect(self.arrastrar_musica) #
Conectar la acción a la función arrastrar_musica
```

Crea el menú de la aplicación del reproductor.

```
def crear_menu(self):
        """
        Crea el menú de la aplicación del reproductor.
        """
        self.menuBar()  # Crear la barra de menú

        menu_archivo = self.menuBar().addMenu("ARCHIVO")  # Crear menú "Archivo"
        menu_archivo.addAction(self.abrir_archivo_musica_action)  # Añadir acción
de abrir carpeta al menú "Archivo"
```

```
        menu_vista = self.menuBar().addMenu("VISTA")  # Crear menú "Vista"
        menu_vista.addAction(self.listar_musica_accion)
        menu_vista.addAction(self.arrastrar_musica_accion)# Añadir acción de
listar música al menú "Vista"
```

Crea un dock para la lista de canciones.

```
def crear_dock(self):
        """
        Crea un dock para la lista de canciones.
        """
        self.dock_list = QDockWidget('LISTAR CANCION POR CARPETA', self)  # Crear
dock para la lista de canciones
        self.dock_list.setWidget(QWidget())  # Establecer un widget vacío como
contenido inicial del dock
        self.songs_list = QListWidget()  # Crear lista de canciones
        self.songs_list.itemSelectionChanged.connect(self.handle_song_selection)
 # Conectar cambio de selección a la función handle_song_selection
        self.dock_list.setWidget(self.songs_list)  # Establecer la lista de
canciones como contenido del dock
        self.addDockWidget(Qt.LeftDockWidgetArea, self.dock_list)  # Añadir el
dock al área izquierda
```

Crea el dock para arrastrar música.

```
def crear_dock_arrastrar(self):
        """
        Crea el dock para arrastrar música.
        """
        self.dock_drag = QDockWidget("ARRASTRAR Y SOLTAR LA CANCION", self)  #
Crea un QDockWidget con el título especificado
        self.dock_drag.setWidget(QWidget())  # Establece un QWidget vacío como el
widget del dock
        self.drag_list = QListWidget()  # Crea un QListWidget para la lista de
canciones arrastradas
        self.drag_list.setAcceptDrops(True)  # Permite que el QListWidget acepte
archivos arrastrados
        self.drag_list.itemDoubleClicked.connect(self.reproducir_musica_lista)  #
Conecta la señal de doble clic en un elemento a la función
reproducir_musica_lista
        self.dock_drag.setWidget(self.drag_list)  # Establece el QListWidget como
el widget del dock
        self.addDockWidget(Qt.RightDockWidgetArea, self.dock_drag)  # Añade el
dock a la área de dock derecha de la ventana principal
        self.dock_drag.hide()  # Oculta el dock inicialmente
```

```
        self.setAcceptDrops(True)  # Permite que la ventana principal acepte
archivos arrastrados
```

Reproduce la música seleccionada de la lista de arrastrar y soltar.

```
 def reproducir_musica_lista(self, item):
        """
        Reproduce la música seleccionada de la lista de arrastrar y soltar.
        """
        file_path = item.data(Qt.UserRole)  # Obtiene la ruta del archivo desde
el elemento de la lista usando el rol de usuario
        self.reproducir_musica(file_path)  # Llama al método reproducir_musica
con la ruta del archivo
```

Reproduce la música desde el archivo especificado.

```
def reproducir_musica(self, file_path):
        """
        Reproduce la música desde el archivo especificado.
        """
        self.player.setMedia(QMediaContent(QUrl.fromLocalFile(file_path)))  #
Establece el medio del reproductor con el archivo local especificado
        self.player.play()  # Inicia la reproducción de la música
        self.song_name_label.setText(os.path.basename(file_path))  # Actualiza la
etiqueta del nombre de la canción con el nombre del archivo
```

Evento de arrastrar que se ejecuta cuando un archivo entra en el área de arrastrar y soltar.

```
def dragEnterEvent(self, event: QDragEnterEvent):
        """
        Evento de arrastrar que se ejecuta cuando un archivo entra en el área de
arrastrar y soltar.
        """
        if event.mimeData().hasUrls():  # Verifica si los datos arrastrados
contienen URLs
            event.acceptProposedAction()  # Acepta la acción propuesta para
permitir el arrastre
```

Evento de soltar que se ejecuta cuando un archivo se suelta en el área de arrastrar y soltar.

```
def dropEvent(self, event: QDropEvent):
        """
        Evento de soltar que se ejecuta cuando un archivo se suelta en el área de arrastrar y soltar.
        """
        if event.mimeData().hasUrls():  # Verifica si los datos soltados contienen URLs
            icono = QIcon("img/caratula_m.jpeg")  # Establecer icono para los elementos de la lista
            for url in event.mimeData().urls():  # Itera sobre las URLs contenidas en los datos soltados
                file_path = url.toLocalFile()  # Convierte la URL en una ruta de archivo local
                if file_path.endswith(('.mp3', '.wav', '.flac')):  # Verifica si el archivo es un tipo de audio soportado
                    item = QListWidgetItem(icono, os.path.basename(file_path))  # Crea un nuevo elemento de lista con el icono y el nombre del archivo
                    item.setData(Qt.UserRole, file_path)  # Almacena la ruta del archivo en el elemento de lista usando el rol de usuario
                    self.drag_list.addItem(item)  # Añade el elemento a la lista de arrastrar y soltar
            event.acceptProposedAction()  # Acepta la acción propuesta para finalizar el evento de soltar
```

Muestra u oculta el dock de la lista de canciones basado en el estado de la acción de listar música.

```
def listar_musica(self):
        """
        Muestra u oculta el dock de la lista de canciones basado en el estado de la acción de listar música.
        """
        if self.listar_musica_accion.isChecked():  # Comprobar si la acción de listar música está seleccionada
            self.dock_list.show()  # Mostrar el dock de la lista de canciones
            self.dock_drag.hide()  # Ocultar el dock de arrastrar música
            self.arrastrar_musica_accion.setChecked(False) # Desmarca la acción de arrastrar música, estableciendo su estado como no seleccionado

        else:
            self.dock_list.hide() # Ocultar el dock de la lista de canciones
```

Muestra u oculta el dock de arrastrar música basado en el estado de la acción de arrastrar música.

```
def arrastrar_musica(self):
        """
        Muestra u oculta el dock de arrastrar música basado en el estado de la
acción de arrastrar música.
        """
        if self.arrastrar_musica_accion.isChecked():  # Comprobar si la acción de
arrastrar música está seleccionada
            self.dock_drag.show()  # Mostrar el dock de arrastrar música
            self.dock_list.hide()  # Ocultar el dock de la lista de canciones
            self.listar_musica_accion.setChecked(False) # Desmarca la acción de
listar música, estableciendo su estado como no seleccionado

        else:
            self.dock_drag.hide()  # Ocultar el dock de arrastrar música
```

Abre un cuadro de diálogo para seleccionar una carpeta de música.

```
def abrir_carpeta_musica(self):
        """
        Abre un cuadro de diálogo para seleccionar una carpeta de música.
        """

        Ubicacion_musica =
QStandardPaths.writableLocation(QStandardPaths.MusicLocation)  # Obtener
ubicación estándar de la música
        self.current_music_folder = QFileDialog.getExistingDirectory(self,
"SELECCIONE UNA CARPETA", Ubicacion_musica)  # Abrir diálogo para seleccionar
carpeta de música

        if not self.current_music_folder:  # Comprobar si no se seleccionó
ninguna carpeta
            return

        self.songs_list.clear()  # Limpiar la lista de canciones
        icono = QIcon("img/caratula_principal.jpeg")  # Establecer icono para los
elementos de la lista
        for archivo in os.listdir(self.current_music_folder):  # Recorrer
archivos en la carpeta seleccionada
            if archivo.endswith(".mp3"):  # Comprobar si el archivo es un MP3
                item = QListWidgetItem(icono, archivo)  # Crear un elemento de
lista con el icono y el nombre del archivo
                self.songs_list.addItem(item)  # Añadir el elemento a la lista de
canciones
```

Maneja la selección de una canción en la lista de canciones.

```
def handle_song_selection(self):
        """
        Maneja la selección de una canción en la lista de canciones.
        """
        seleccion_cancion = self.songs_list.selectedItems()  # Obtener los
elementos seleccionados en la lista de canciones
        if seleccion_cancion:  # Comprobar si hay algún elemento seleccionado
            nombre_cancion = seleccion_cancion[0].text()  # Obtener el nombre de
la canción seleccionada
            self.song_name_label.setText(nombre_cancion)  # Establecer el nombre
de la canción en la etiqueta
            song_folder_path = os.path.join(self.current_music_folder,
nombre_cancion)  # Obtener la ruta completa de la canción
            source = QMediaContent(QUrl.fromLocalFile(song_folder_path))  # Crear
un objeto QMediaContent con la ruta de la canción
            self.player.setMedia(source)  # Establecer la canción en el
reproductor
            self.actualizar_estilo_boton_play()
            self.player.play()  # Reproducir la canción
            self.actualizar_estilo_boton_play()
            self.is_paused = False  # Establecer el estado de pausa a False
            self.current_song_index = self.songs_list.currentRow()  # Establecer
el índice de la canción actual
```

Maneja la selección de una canción en la lista de canciones arrastradas.

```
def handle_drag_song_selection(self):
        """
        Maneja la selección de una canción en la lista de canciones arrastradas.
        """
        seleccion_cancion = self.drag_list.selectedItems()  # Obtener los
elementos seleccionados en la lista de canciones arrastradas
        if seleccion_cancion:  # Comprobar si hay algún elemento seleccionado
            file_path = seleccion_cancion[0].data(Qt.UserRole)  # Obtener la ruta
del archivo de la canción seleccionada
            self.song_name_label.setText(os.path.basename(file_path))  #
Establecer el nombre de la canción en la etiqueta
            source = QMediaContent(QUrl.fromLocalFile(file_path))  # Crear un
objeto QMediaContent con la ruta de la canción
            self.player.setMedia(source)  # Establecer la canción en el
reproductor
            self.actualizar_estilo_boton_play()
            self.player.play()  # Reproducir la canción
            self.actualizar_estilo_boton_play()
```

```
            self.is_paused = False  # Establecer el estado de pausa a False
            self.current_song_index = self.drag_list.currentRow()  # Establecer
el índice de la canción actual
        else:
            self.current_song_index = -1  # Ninguna canción seleccionada
```

Reproduce la canción actual desde el principio.

```
def repetir(self):
        """
        Reproduce la canción actual desde el principio.
        """
        self.player.setPosition(0)  # Reiniciar la canción al inicio
        self.player.play()  # Reproducir la canción
```

Retrocede a la canción anterior en la lista.

```
def retroceder(self):
        """
        Retrocede a la canción anterior en la lista.
        """
        if self.dock_drag.isVisible():  # Comprueba si el dock de arrastrar y
soltar está visible
            if self.current_song_index > 0:  # Comprueba si hay una canción
anterior en la lista
                self.current_song_index -= 1  # Decrementa el índice de la
canción actual para retroceder
                self.drag_list.setCurrentRow(self.current_song_index)  #
Establece la fila actual de la lista de arrastrar y soltar al nuevo índice
                self.handle_drag_song_selection()  # Maneja la selección de la
canción en la lista de arrastrar y soltar
        else:
            if self.current_song_index > 0:  # Comprueba si hay una canción
anterior en la lista
                self.current_song_index -= 1  # Decrementa el índice de la
canción actual para retroceder
                self.songs_list.setCurrentRow(self.current_song_index)  #
Establece la fila actual de la lista de canciones al nuevo índice
                self.handle_song_selection()  # Maneja la selección de la canción
en la lista de canciones
```

Alterna entre reproducir y pausar la canción actual.

```
def play_pause(self):
        """
        Alterna entre reproducir y pausar la canción actual.
        """
        self.actualizar_estilo_boton_play()
        if self.player:
            self.actualizar_estilo_boton_play()
            if self.player.state() == QMediaPlayer.PlayingState:  # Comprobar si
el reproductor está en estado de reproducción
                self.actualizar_estilo_boton_play()
                self.player.pause()  # Pausar la reproducción
                self.is_paused = True  # Establecer el estado de pausa a True
            else:
                self.player.play()  # Reproducir la canción
                self.is_paused = False  # Establecer el estado de pausa a False

            self.actualizar_estilo_boton_play()  # Actualizar el estilo del botón
de play/pausa
```

Actualiza el estilo del botón de play/pausa según el estilo actual de la aplicación.

```
def actualizar_estilo_boton_play(self):
        """
        Actualiza el estilo del botón de play/pausa según el estilo actual de la
aplicación.
        """
        estilo_actual = self.estilo_combo.currentIndex()  # Obtener el índice del
estilo actual seleccionado en el combo box
        estilos = {  # Diccionario que mapea los índices de los estilos a los
estilos específicos para los estados de play y pausa
            0: {  # Estilo principal
                "play": "QPushButton#boton_play { background-color: gray; image:
url('img/play.png'); border-radius: 35px; padding: 7px; }",
                "pause": "QPushButton#boton_play { background-color: gray; image:
url('img/pausa.png'); border-radius: 45px; padding: 7px; }"
            },
            1: {  # Estilo retro
                "play": "QPushButton#boton_play { background-color: rgba(255,
165, 0, 0.3); image: url('img/retro/play.png'); border-radius: 35px; padding:
7px; }",
                "pause": "QPushButton#boton_play { background-color: rgba(255,
165, 0, 0.3); image: url('img/retro/pausa.png'); border-radius: 45px; padding:
7px; }"
            },
```

```
            2: {  # modo dj
                "play": "QPushButton#boton_play { background-color: rgba(0, 255,
0, 0.1);  image: url('img/moderno/play.png'); border-radius: 35px; padding:
7px;  }",
                "pause": "QPushButton#boton_play { background-color: rgba(0, 255,
0, 0.1);  image: url('img/moderno/pausa.png'); border-radius: 45px; padding:
7px;  }"
            }
        }

        estilo_actual = self.estilo_combo.currentIndex()  # Obtener el índice del
estilo actual
        if self.is_paused:
            self.boton_play.setStyleSheet(estilos[estilo_actual]["play"])  #
Establecer el estilo para el estado de pausa
        else:
            self.boton_play.setStyleSheet(estilos[estilo_actual]["pause"])  #
Establecer el estilo para el estado de reproducción
```

Reproduce una canción aleatoria de la lista.

```
def aleatorio(self):
        """Reproduce
        Reproduce una canción aleatoria de la lista.
        """
        if self.dock_drag.isVisible():  # Comprueba si el dock de arrastrar y
soltar está visible
            if self.drag_list.count() > 0:  # Verifica si hay al menos una
canción en la lista de arrastrar y soltar
                self.current_song_index = random.randint(0,
self.drag_list.count() - 1)  # Selecciona un índice aleatorio dentro del rango de
la lista de arrastrar y soltar
                self.drag_list.setCurrentRow(self.current_song_index)  #
Establece la fila actual de la lista de arrastrar y soltar al índice aleatorio
                self.handle_drag_song_selection()  # Maneja la selección de la
canción en la lista de arrastrar y soltar
        else:
            if self.songs_list.count() > 0:  # Verifica si hay al menos una
canción en la lista de canciones
                self.current_song_index = random.randint(0,
self.songs_list.count() - 1)  # Selecciona un índice aleatorio dentro del rango
de la lista de canciones
                self.songs_list.setCurrentRow(self.current_song_index)  #
Establece la fila actual de la lista de canciones al índice aleatorio
                self.handle_song_selection()  # Maneja la selección de la canción
en la lista de canciones
```

Establece la posición de reproducción al mover la barra de progreso.

```
def set_position(self, position):
        """
        Establece la posición de reproducción al mover la barra de progreso.
        """
        self.player.setPosition(position)  # Establecer la posición de
reproducción en el reproductor
```

Establece la posición de reproducción al mover la barra de progreso.

```
def slider_pressed(self):
        """
        Pausa la reproducción al presionar la barra de progreso.
        """
        self.player.pause()  # Pausar la reproducción cuando se presiona el
slider
```

Reanuda la reproducción al soltar la barra de progreso.

```
def slider_released(self):
        """
        Reanuda la reproducción al soltar la barra de progreso.
        """
        self.player.play()  # Reproducir la canción cuando se libera el slider
```

Establece el volumen del reproductor de música.

```
def set_volume(self, volume):
        """
        Establece el volumen del reproductor de música.
        """
        self.player.setVolume(volume)  # Establecer el volumen del reproductor
```

Actualiza la duración de la barra de progreso según la duración de la canción.

```
def actualizar_duracion(self, duration):
        """
        Actualiza la duración de la barra de progreso según la duración de la
canción.
        """
        self.progress_bar.setRange(0, duration)  # Establecer el rango de la
barra de progreso
        self.total_time_label.setText(self.formato_tiempo(duration))  #
Actualizar la etiqueta del tiempo total
```

Convierte milisegundos en un formato de tiempo legible (mm:ss).

```
def formato_tiempo(self, ms):
        """
        Convierte milisegundos en un formato de tiempo legible (mm:ss).
        """
        seconds = (ms // 1000) % 60  # Calcular segundos
        minutes = (ms // (1000 * 60)) % 60  # Calcular minutos
        hours = (ms // (1000 * 60 * 60)) % 24  # Calcular horas
        if hours > 0:  # Si hay horas
            return f"{hours:02}:{minutes:02}:{seconds:02}"  # Devolver tiempo en
formato hh:mm:ss
        else:
            return f"{minutes:02}:{seconds:02}"  # Devolver tiempo en formato
mm:ss
```

Genera la pestaña de configuración.

```
def generar_configuracion_tab(self):
        """
        Genera la pestaña de configuración.
        """
        configuracion_v_box = QVBoxLayout()  # Crear layout vertical para la
configuración
        configuracion_v_box.setAlignment(Qt.AlignTop)  # Alinear al inicio
        self.configuracion_container.setStyleSheet("background-color: rgba(0, 0,
0, 100); border-radius: 10px; padding: 20px;")# Establece un estilo para el
container del reproductor

        # Encabezado
        self.header_label = QLabel("Configuración del Reproductor")
        self.header_label.setStyleSheet("color: white; font-size: 20px; margin-
bottom: 20px;")
        configuracion_v_box.addWidget(self.header_label)  # Añadir el encabezado
al layout de configuración

        # Estilo de interfaz
        self.estilo_label = QLabel("Selecciona el estilo del reproductor:")  #
Crear etiqueta para seleccionar el estilo
        self.estilo_label.setStyleSheet("color: white; font-size: 16px;")  #
Establecer el estilo de la etiqueta
        self.estilo_combo = QComboBox()  # Crear combo box para seleccionar el
estilo
        self.estilo_combo.addItem("Estilo Principal")  # Añadir opción de estilo
1
        self.estilo_combo.addItem("Estilo Retro")  # Añadir opción de estilo 2
        self.estilo_combo.addItem("Modo DJ")  # Añadir opción de estilo 3
```

```
        self.estilo_combo.currentIndexChanged.connect(self.cambiar_estilo)  # Conectar el cambio de selección a la función cambiar_estilo
        self.estilo_combo.setStyleSheet("""
            QComboBox {
                background-color: rgba(0, 0, 0, 100);
                color: white;
                border: 1px solid #555;
                border-radius: 5px;
                padding: 5px;
            }
           QComboBox::drop-down {
               subcontrol-origin: padding;
                subcontrol-position: top right;
                width: 15px;
                border-left: 1px solid #555;
                background-color: rgba(0, 0, 0, 100);
                image: url(img/a.png);
            }


            QComboBox QAbstractItemView {
                background-color: #444;
                color: white;
                selection-background-color: #555;
            }
        """)

        configuracion_v_box.addWidget(self.estilo_label)  # Añadir la etiqueta al layout de configuración
        configuracion_v_box.addWidget(self.estilo_combo)  # Añadir el combo box al layout de configuración


        self.configuracion_container.setLayout(configuracion_v_box)  # Establecer el layout para el contenedor de configuración
```

Cambia el estilo de la interfaz según la selección.

```
def cambiar_estilo(self, index):
        """
        Cambia el estilo de la interfaz según la selección.
        """
        estilos = [self.estilo_1, self.estilo_2, self.estilo_3]  # Lista de estilos
        estilo_seleccionado = estilos[index] if index < len(estilos) else self.estilo_1  # Obtener el estilo seleccionado, o estilo_1 si index está fuera de rango
```

```
        self.setStyleSheet(estilo_seleccionado())  # Establecer el estilo en la aplicación
        self.actualizar_estilo_boton_play()  # Actualizar el estilo del botón de play/pausa

        # Cambiar la carátula de la música según el estilo seleccionado
        caratula_seleccionada = estilo_seleccionado(return_caratula=True)
        pixmap = QPixmap(caratula_seleccionada).scaled(400, 350)
        self.fondo_imagen.setPixmap(pixmap)
```

Define el estilo 1.

```
def estilo_1(self, return_caratula=False):
        if return_caratula:
            return "img/caratula_m.jpeg"
        """
        Define el estilo 1.
        """
        return """
       QMainWindow {
            background-image: url(img/fon.jpeg);
            background-position: center;
            background-repeat: no-repeat;
            background-size: cover;
        }
        QDockWidget {
           background-color: rgba(0, 0, 0, 0.8);
           color: white;
           font-weight: bold;
        }
        QListWidget {
          background-color: rgba(0, 0, 0, 0.8);
          color: white;
        }
        QTabWidget::pane {
         background-color: rgba(255, 255, 255, 0.1);
         border: 0;
        }
        QTabBar::tab {
           background-color: rgba(255, 255, 255, 0.1);
           color: white;
           font-weight: bold;
        }

        /* Estilos de botones con fondo gris transparente y bordes redondeados */
        QPushButton#boton_repetir {
```

```
          background-color: rgba(128, 128, 128, 150); /* Gris transparente */
          border: none; /* Remover cualquier borde existente */
          border-radius: 25px; /* Bordes redondeados */
          image: url('img/repetir.png'); /* Imagen del botón */
          padding: 7px; /* Relleno */
        }
        QPushButton#boton_retroceder {
          background-color: rgba(128, 128, 128, 150); /* Gris transparente */
          border: none; /* Remover cualquier borde existente */
          border-radius: 25px; /* Bordes redondeados */
          image: url('img/retroceder.png'); /* Imagen del botón */
          padding: 7px; /* Relleno */
        }
        QPushButton#boton_play {
          background-color: rgba(128, 128, 128, 150); /* Gris transparente */
          border: none; /* Remover cualquier borde existente */
          border-radius: 25px; /* Bordes redondeados */
          image: url('img/play.png'); /* Imagen del botón */
          padding: 7px; /* Relleno */
        }
        QPushButton#boton_avanzar {
           background-color: rgba(128, 128, 128, 150); /* Gris transparente */
           border: none; /* Remover cualquier borde existente */
           border-radius: 25px; /* Bordes redondeados */
           image: url('img/avanzar.png'); /* Imagen del botón */
           padding: 7px; /* Relleno */
        }
        QPushButton#boton_aleatorio {
            background-color: rgba(128, 128, 128, 150); /* Gris transparente */
            border: none; /* Remover cualquier borde existente */
            border-radius: 25px; /* Bordes redondeados */
            image: url('img/repetir.png'); /* Imagen del botón */
            padding: 7px; /* Relleno */
        }
        /* Efecto visual al presionar los botones */
        QPushButton#boton_repetir:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }
        QPushButton#boton_retroceder:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }
        QPushButton#boton_retroceder_a:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
```

```
        }
        QPushButton#boton_play:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }
        QPushButton#boton_pausa:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }
        QPushButton#boton_avanzar:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }
        QPushButton#boton_aleatorio:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }
        QStatusBar {
            background-color: rgba(78, 78, 78, 255);
            color: white;
            font-weight: bold;
        }
        QMenuBar {
            background-color: rgba(29, 29, 29, 255);
            color: white;
            font-weight: bold;
        }
        QMenuBar::item {
            background-color: rgba(29, 29, 29, 255);
            color: white;
            font-weight: bold;
        }
        QMenu {
            background-color: rgba(29, 29, 29, 255);
            color: white;
            font-weight: bold;
        }
        QMenu::item:selected {
            background-color: rgba(78, 78, 78, 255);
        }
        QMenu::item:pressed {
            background-color: gray;
        }
        QMenuBar::item:pressed {
            background-color: gray;
        }
```

```
        QTabWidget::pane::pressed {
            background-color: gray;
        }
        QTabBar::tab::pressed {
            background-color: gray;
        }
        /* Barra de progreso moderna */
        QSlider::groove:horizontal {
            border: 1px solid #999;
            background: #444;
            height: 10px;
            border-radius: 5px;
        }
        QSlider::sub-page:horizontal {
            background: #3b5998;
            border: 1px solid #555;
            height: 10px;
            border-radius: 5px;
        }
        QSlider::add-page:horizontal {
            background: #777;
            border: 1px solid #999;
            height: 10px;
            border-radius: 5px;
        }
        QSlider::handle:horizontal {
            background: #ffcc00;
            border: 1px solid #999;
            width: 20px;
            height: 20px;
            margin-top: -5px;
            margin-bottom: -5px;
            border-radius: 10px;
        }
        /* Estilo  barra de volumen */
        QSlider::groove:vertical {
            border: 1px solid #999;
            background: #444;
            width: 10px;
            border-radius: 5px;
        }
        QSlider::sub-page:vertical {
            background: #3b5998;
            border: 1px solid #555;
            width: 10px;
            border-radius: 5px;
```

```
        }
        QSlider::add-page:vertical {
            background: #777;
            border: 1px solid #999;
            width: 10px;
            border-radius: 5px;
        }
        QSlider::handle:vertical {
            background: #ffcc00;
            border: 1px solid #999;
            width: 20px;
            height: 20px;
            margin-left: -5px;
            margin-right: -5px;
            border-radius: 10px;
        }
        """
```

Define el estilo 2.

```
def estilo_2(self, return_caratula=False):
        if return_caratula:
            return "img/retro/fondo_retro.jpg"

        """
        Define el estilo 2.
        """
        return """
        QMainWindow {
            background-image: url(img/retro/retro.jpg);
            background-position: center;
            background-repeat: no-repeat;
            background-size: cover;
        }


        QDockWidget {
           background-color: rgba(0, 0, 0, 0.8);
           color: white;
           font-weight: bold;
        }

        QListWidget {
          background-color: rgba(0, 0, 0, 0.8);
          color: white;
        }
```

```
QTabWidget::pane {
 background-color: rgba(139, 69, 19, 0.3);
 border-radius: 20px;
}

QTabBar::tab {
   background-color: rgba(255, 165, 0, 0.3);
   color: white;
   font-weight: bold;

}

/* Estilos de botones con fondo gris transparente y bordes redondeados */
QPushButton#boton_repetir {
  background-color: rgba(255, 165, 0, 0.3); /* Gris transparente */
  border: none; /* Remover cualquier borde existente */
  border-radius: 25px; /* Bordes redondeados */
  image: url('img/retro/repetir.png'); /* Imagen del botón */
  padding: 7px; /* Relleno */
}

QPushButton#boton_retroceder {
  background-color: rgba(255, 165, 0, 0.3); /* Gris transparente */
  border: none; /* Remover cualquier borde existente */
  border-radius: 25px; /* Bordes redondeados */
  image: url('img/retro/retroceder.png'); /* Imagen del botón */
  padding: 7px; /* Relleno */
}

QPushButton#boton_play {
  background-color: rgba(255, 165, 0, 0.3); /* Gris transparente */
  border: none; /* Remover cualquier borde existente */
  border-radius: 25px; /* Bordes redondeados */
  image: url('img/retro/play.png'); /* Imagen del botón */
  padding: 7px; /* Relleno */
}

QPushButton#boton_avanzar {
   background-color: rgba(255, 165, 0, 0.3); /* Gris transparente */
   border: none; /* Remover cualquier borde existente */
   border-radius: 25px; /* Bordes redondeados */
   image: url('img/retro/avanzar.png'); /* Imagen del botón */
   padding: 7px; /* Relleno */
}

QPushButton#boton_aleatorio {
```

```
            background-color: rgba(255, 165, 0, 0.3); /* Gris transparente */
            border: none; /* Remover cualquier borde existente */
            border-radius: 25px; /* Bordes redondeados */
            image: url('img/retro/repetir.png'); /* Imagen del botón */
            padding: 7px; /* Relleno */
        }

        /* Efecto visual al presionar los botones */
        QPushButton#boton_repetir:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }

        QPushButton#boton_retroceder:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }

        QPushButton#boton_retroceder_a:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }

        QPushButton#boton_play:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }

        QPushButton#boton_pausa:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }

        QPushButton#boton_avanzar:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }

        QPushButton#boton_aleatorio:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }

        QStatusBar {
            background-color: rgba(78, 78, 78, 255);
            color: white;
```

```
            font-weight: bold;
        }

        QMenuBar {
            background-color: rgba(29, 29, 29, 255);
            color: white;
            font-weight: bold;
        }

        QMenuBar::item {
            background-color: rgba(29, 29, 29, 255);
            color: white;
            font-weight: bold;
        }

        QMenu {
            background-color: rgba(29, 29, 29, 255);
            color: white;
            font-weight: bold;
        }

        QMenu::item:selected {
            background-color: rgba(78, 78, 78, 255);
        }

        QMenu::item:pressed {
            background-color: gray;
        }

        QMenuBar::item:pressed {
            background-color: gray;
        }

        QTabWidget::pane::pressed {
            background-color: gray;
        }

        QTabBar::tab::pressed {
            background-color: gray;
        }

        /* Barra de progreso moderna */
        QSlider::groove:horizontal {
            border: 1px solid #999;
            background: #444;
            height: 10px;
```

```
        border-radius: 5px;
    }

    QSlider::sub-page:horizontal {
        background: #3b5998;
        border: 1px solid #555;
        height: 10px;
        border-radius: 5px;
    }

    QSlider::add-page:horizontal {
        background: #777;
        border: 1px solid #999;
        height: 10px;
        border-radius: 5px;
    }

    QSlider::handle:horizontal {
        background-color: #FFA500;
        border: 1px solid #999;
        width: 20px;
        height: 20px;
        margin-top: -5px;
        margin-bottom: -5px;
        border-radius: 10px;
    }

    /* Estilo  barra de volumen */
    QSlider::groove:vertical {
        border: 1px solid #999;
        background: #444;
        width: 10px;
        border-radius: 5px;
    }

    QSlider::sub-page:vertical {
        background: #3b5998;
        border: 1px solid #555;
        width: 10px;
        border-radius: 5px;
    }

    QSlider::add-page:vertical {
        background: #777;
        border: 1px solid #999;
        width: 10px;
```

```
            border-radius: 5px;
        }

        QSlider::handle:vertical {
            background-color: rgba(255, 165, 0, 0.3);
            border: 1px solid #999;
            width: 20px;
            height: 20px;
            margin-left: -5px;
            margin-right: -5px;
            border-radius: 10px;
        }
        QPushButton#icono_volumen {
            background-color: #FFA500;
            border: none;
            border-radius: 25px;
            padding: 10px;
        }
        """
```

Define el estilo 3.

```
def estilo_3(selfself, return_caratula=False):
        if return_caratula:
            return "img/caratula_principal.jpeg"
        """
        Define el estilo 3.
        """
        return """
        QMainWindow {
            background-image: url(img/moderno/dj.jpg);
            background-position: center;
            background-repeat: no-repeat;
            background-size: cover;
        }


        QDockWidget {
           background-color: rgba(0, 0, 0, 0.8);
           color: white;
           font-weight: bold;
        }

        QListWidget {
          background-color: rgba(0, 0, 0, 0.8);
          color: white;
          border-radius: 20px;
```

```
}

QTabWidget::pane {
 background-color: rgba(255, 255, 255, 0.1);
 border-radius: 20px;
}

QTabBar::tab {
   background-color: rgba(0, 255, 0, 0.1);
   color: white;
   font-weight: bold;
}

/* Estilos de botones con fondo gris transparente y bordes redondeados */
QPushButton#boton_repetir {
  background-color: rgba(0, 255, 0, 0.1); /* Gris transparente */
  border: none; /* Remover cualquier borde existente */
  border-radius: 25px; /* Bordes redondeados */
  image: url('img/moderno/repetir.png'); /* Imagen del botón */
  padding: 7px; /* Relleno */

}

QPushButton#boton_retroceder {
  background-color: rgba(0, 255, 0, 0.1); /* Gris transparente */
  border: none; /* Remover cualquier borde existente */
  border-radius: 25px; /* Bordes redondeados */
  image: url('img/moderno/retroceder.png'); /* Imagen del botón */
  padding: 7px; /* Relleno */

}

QPushButton#boton_play {
  background-color: rgba(0, 255, 0, 0.1); /* Gris transparente */
  border: none; /* Remover cualquier borde existente */
  border-radius: 25px; /* Bordes redondeados */
  image: url('img/moderno/play.png'); /* Imagen del botón */
  padding: 7px; /* Relleno */


}

QPushButton#boton_avanzar {
   background-color: rgba(0, 255, 0, 0.1); /* Gris transparente */
   border: none; /* Remover cualquier borde existente */
   border-radius: 25px; /* Bordes redondeados */
```

```
            image: url('img/moderno/avanzar.png'); /* Imagen del botón */
            padding: 7px; /* Relleno */

        }

        QPushButton#boton_aleatorio {
            background-color: rgba(0, 255, 0, 0.1); /* Gris transparente */
            border: none; /* Remover cualquier borde existente */
            border-radius: 25px; /* Bordes redondeados */
            image: url('img/moderno/aleatorio.png'); /* Imagen del botón */
            padding: 7px; /* Relleno */

        }

        /* Efecto visual al presionar los botones */
        QPushButton#boton_repetir:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }

        QPushButton#boton_retroceder:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }

        QPushButton#boton_retroceder_a:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }

        QPushButton#boton_play:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }

        QPushButton#boton_pausa:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }

        QPushButton#boton_avanzar:pressed {
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }

        QPushButton#boton_aleatorio:pressed {
```

```
            background-color: rgba(128, 128, 128, 200); /* Cambia la
transparencia para indicar el clic */
        }

        QStatusBar {
            background-color: rgba(78, 78, 78, 255);
            color: white;
            font-weight: bold;
        }

        QMenuBar {
            background-color: rgba(29, 29, 29, 255);
            color: white;
            font-weight: bold;
        }

        QMenuBar::item {
            background-color: rgba(29, 29, 29, 255);
            color: white;
            font-weight: bold;
        }

        QMenu {
            background-color: rgba(29, 29, 29, 255);
            color: white;
            font-weight: bold;
        }

        QMenu::item:selected {
            background-color: rgba(78, 78, 78, 255);
        }

        QMenu::item:pressed {
            background-color: gray;
        }

        QMenuBar::item:pressed {
            background-color: gray;
        }

        QTabWidget::pane::pressed {
            background-color: gray;
        }

        QTabBar::tab::pressed {
            background-color: gray;
```

```
        }

        /* Barra de progreso moderna */
        QSlider::groove:horizontal {
            border: 1px solid #999;
            background: #444;
            height: 10px;
            border-radius: 5px;
        }

        QSlider::sub-page:horizontal {
            background: #3b5998;
            border: 1px solid #555;
            height: 10px;
            border-radius: 5px;
        }

        QSlider::add-page:horizontal {
            background: #777;
            border: 1px solid #999;
            height: 10px;
            border-radius: 5px;
        }

        QSlider::handle:horizontal {
            background: #ffcc00;
            border: 1px solid #999;
            width: 20px;
            height: 20px;
            margin-top: -5px;
            margin-bottom: -5px;
            border-radius: 10px;
        }

        /* Estilo  barra de volumen */
        QSlider::groove:vertical {
            border: 1px solid #999;
            background: #444;
            width: 10px;
            border-radius: 5px;
        }

        QSlider::sub-page:vertical {
            background: #3b5998;
            border: 1px solid #555;
            width: 10px;
```

```
            border-radius: 5px;
        }

        QSlider::add-page:vertical {
            background: #777;
            border: 1px solid #999;
            width: 10px;
            border-radius: 5px;
        }

        QSlider::handle:vertical {
            background: #ffcc00;
            border: 1px solid #999;
            width: 20px;
            height: 20px;
            margin-left: -5px;
            margin-right: -5px;
            border-radius: 10px;
        }
        """
```

Punto de entrada principal del programa.

```
if __name__ == '__main__':
    """
    Punto de entrada principal del programa.
    """

    app = QApplication(sys.argv)  # Crear la aplicación
    app.setAttribute(Qt.AA_UseHighDpiPixmaps)
    window = MusicPlayer()  # Crear la ventana del reproductor de música
    window.show()  # Mostrar la ventana
    sys.exit(app.exec_())  # Ejecutar el ciclo de eventos de la aplicación
```

CONCLUSION

En el transcurso de esta investigación, se ha explorado exhaustivamente el potencial de la biblioteca PyQt en el desarrollo de un reproductor de música, destacando su versatilidad, robustez y eficiencia en la creación de interfaces gráficas de usuario (GUI) dinámicas y funcionales.

Este reproductor de música no solo representa el éxito de la investigación, sino también el inicio de nuevas posibilidades y oportunidades para el desarrollo de aplicaciones innovadoras y centradas en el usuario utilizando la biblioteca PyQt. Con un enfoque continuo en la mejora y la innovación, el futuro del desarrollo de software multimedia con PyQt es prometedor y emocionante.

Webgrafía

- https://www.youtube.com/watch?v=wWN5TZsfFUA&t=1090s
- https://chatgpt.com
- https://github.com/MagnoEfren