



Trabajo Fin de Grado

Musy



Autor: Beltrán González Martos

Tutor: Héctor Ángeles Borrás

Desarrollo de Aplicaciones Multiplataforma

Fecha: 9 de junio de 2025

Resumen

Este proyecto aborda la ausencia de reproductores de musica offline modernos además de abordar el alto consumo de recursos en aplicaciones web, proponiendo una solución basada en tauri (Rust + Angular) para garantizar eficiencia. Se desarrolló una aplicación de escritorio compatible con Linux, MacOS y Windows, priorizando la optimización de memoria y la experiencia de usuario. El resultado es una aplicación escalable con arquitectura modular para futuras extensiones, validando así el potencial de Tauri en aplicaciones de escritorio.

Palabras clave: Tauri, Rust, Angular, reproductor de música, rendimiento, UI moderna.

Abstract

This project addresses both the lack of modern offline music players and the high resource consumption in web applications by proposing an efficient solution based on Tauri (Rust + Angular). A cross-platform desktop application was developed for Linux, MacOS, and Windows, with the particular emphasis on memory optimization and user experience. The result is a scalable application with modular architecture for future extensions, demonstrating Tauri's potential for desktop applications.

Keywords: Tauri, Rust, Angular, music player, performance, modern UI

Índice

1. Introducción	1
2. Justificación del tema elegido	1
3. Objetivos	1
4. Metodología	1
5. Tecnologías y herramientas usadas en el proyecto	2
5.1. Razones	2
6. Despliegue y pruebas	3
6.1. Despliegue de la aplicación	3
6.2. Pruebas realizadas	3
6.2.1. Pruebas de integración	3
6.2.2. Pruebas de usabilidad	4
7. Conclusiones	4
7.1. Logros principales	4
7.2. Dificultades clave	5
7.3. Valoración global	5

Índice de figuras

1. Rendimiento de la aplicación en MacOS	4
--	---

Índice de cuadros

1. Introducción

Con el aumento de precios y la inclusión de anuncios en las suscripciones premium de plataformas multimedia como Spotify o iTunes, se prevé un resurgimiento de la reproducción local de música. Este proyecto tiene como objetivo ofrecer una alternativa moderna, intuitiva y sencilla frente a las aplicaciones de reproducción local existentes.

2. Justificación del tema elegido

Aunque existen alternativas a las aplicaciones de reproducción en streaming, resulta difícil encontrar una que combine una interfaz moderna, atención al detalle y código abierto (open-source). Por ello, se ha desarrollado esta propuesta, centrada en una aplicación de reproducción local con una interfaz similar a Spotify o iTunes, lo que facilitará la transición de nuevos usuarios.

3. Objetivos

El objetivo principal es desarrollar una aplicación multiplataforma basada en tecnologías web, eficiente en la gestión de recursos del sistema y fácil de usar. Adicionalmente, se busca que el proyecto sirva como aprendizaje en el uso de Angular y Rust.

4. Metodología

Para la realización del proyecto se empleará la metodología ágil SCRUM. En este caso, el autor asumirá los roles de product owner, scrum master y development team. Las tareas serán asignadas diariamente, y se realizará un seguimiento continuo del progreso.

5. Tecnologías y herramientas usadas en el proyecto

Este proyecto está desarrollado mediante un conjunto de tecnologías modernas distribuidas en tres capas principales:

- **Frontend:** Angular 17, Tailwind CSS V4
- **Backend:** Rust, Tauri 2.0
- **Base de Datos:** SQLite

5.1. Razones

La selección de tecnologías para este proyecto se ha basado en los siguientes criterios técnicos y de eficiencia:

- **Angular 17:** Se ha elegido este framework por su arquitectura basada en componentes, que favorece la escalabilidad y mantenibilidad del código. Además, se ha considerado relevante la oportunidad de explorar alternativas a React, ampliando así el conocimiento en ecosistemas frontend modernos.
- **Tailwind CSS v4:** Se ha optado por esta herramienta debido a su eficiencia en el desarrollo de interfaces responsive, así como a la familiaridad previa con su paradigma utility-first, lo que permite agilizar el proceso de diseño.
- **Rust:** Se ha seleccionado este lenguaje por su rendimiento optimizado y seguridad. Además, representa una valiosa oportunidad de aprendizaje de un lenguaje de programación de bajo nivel.
- **Tauri:** Se ha preferido sobre alternativas como Electron debido a su menor consumo de recursos y mejor rendimiento en aplicaciones de escritorio multiplataforma.
- **SQLite:** Se ha implementado este sistema de gestión de bases de datos por su ligereza, y adecuación a los requisitos del proyecto.

El entorno de desarrollo está configurado en Visual Studio Code, utilizando extensiones específicas para cada tecnología mencionada, lo que garantiza un flujo de trabajo eficiente.

6. Despliegue y pruebas

En este apartado se detallan los pasos seguidos para el despliegue de la aplicación de escritorio, así como las pruebas realizadas para garantizar su correcto funcionamiento.

6.1. Despliegue de la aplicación

Se elaboró un proceso de empaquetado y distribución de la aplicación utilizando Tauri, que permite generar ejecutables multiplataforma (Windows, MacOS, Linux). Para ello:

- Se utilizó el sistema de bundling de Tauri para empaquetar los recursos frontend (Angular) junto al backend (Rust).
- Se generaron instaladores específicos para diferentes sistemas operativos:
 - **.AppImage** para Linux
 - **.exe** para Windows
 - **.dmg** para MacOS

6.2. Pruebas realizadas

Con el objetivo de validar la funcionalidad y estabilidad de la aplicación, se llevaron a cabo las siguientes pruebas:

6.2.1. Pruebas de integración

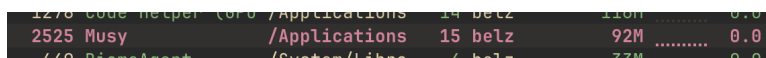
- Se comprobó la comunicación entre el frontend y el backend, asegurando que las llamadas desde Angular a Rust (a través de Tauri) funcionaban correctamente.
- Se testearon casos como la carga de archivos de audio, reproducción en diferentes formatos y manejo de errores.

6.2.2. Pruebas de usabilidad

- Se realizaron test con usuarios reales para evaluar la experiencia de uso (UX), recogiendo feedback sobre la interfaz y la fluidez de la reproducción musical.
- Se analizaron posibles cuellos de botella en el rendimiento, especialmente al manejar playlist con muchas canciones o archivos de alta resolución.

La aplicación fue testeada en los tres sistemas operativos principales (Windows, MacOS y Linux), aunque el desarrollo se centró principalmente en las versiones para MacOS y Linux.

El rendimiento de la aplicación resulta significativamente mejorado al aprovechar las características nativas de Tauri, en comparación con soluciones basadas en Electron.



A screenshot of the macOS Activity Monitor window. The table lists several processes. The process 'Musy' is highlighted in red, showing it is using 15% of the CPU and 92M of memory. Other processes like 'code helper' and 'BionicAgent' are also visible.

Process Name	Path	CPU	Memory	Other
code helper	/Applications	14%	110M	0.0
2525 Musy	/Applications	15%	92M	0.0
669 BionicAgent	/System/Library	4%	33M	0.0

Figura 1: Rendimiento de la aplicación en MacOS

7. Conclusiones

7.1. Logros principales

Se ha desarrollado con éxito un reproductor de música multiplataforma (Linux, MacOS y Windows) utilizando Tauri, que cumple con los objetivos principales de:

- Optimización de recursos (Mucho menor consumo de ram respecto a otras webapps basadas en Electron).
- Compatibilidad de formatos de audio (FLAC, MP3, etc...).
- Arquitectura modular para el desarrollo futuro.

7.2. Dificultades clave

El principal desafío técnico fue la curva de aprendizaje asociada a Rust, lenguaje de programación de sistemas con el que no se contaba experiencia previa. Esta dificultad se resolvió mediante el estudio de documentación oficial, tutoriales especializados y la implementación de pruebas piloto. Adicionalmente, fue necesario adaptar los conocimientos existentes en desarrollo web al framework Angular, cuyo paradigma de componentes requirió un período de adaptación.

7.3. Valoración global

A pesar de las dificultades, el proyecto valida el potencial de Tauri para aplicaciones de audio eficientes, ofreciendo un rendimiento superior al de frameworks tradicionales. La escalabilidad de la arquitectura permite añadir funcionalidades como cambios en la apariencia en futuras iteraciones.