FACULTAD DE INGENIERIA
Universidad de Buenos Aires

El Fogón

Sistema distribuido para músicos con estado compartido del compás, la canción y el repertorio

Segundo Cuatrimestre de 2025

Tutor: Diego Montaldo

Luis Waldman

Padrón: 79279

Índice

Palabras Clave

- Web
- Vue.js
- WebSocket
- WebRTC
- Diseño responsivo
- Golang
- NUnit
- Playwright
- Sincronización
- Compensación de *jitter*
- Música
- Letras
- Acordes
- Partituras

Abstracto

Español

Fogón es un sistema distribuido orientado a facilitar la práctica musical y la sincronización entre músicos. A cada uno le ofrece, desde una aplicación web progresiva, vistas para su instrumento: letras, acordes o partituras. Permite crear listas y editar canciones. También, crear sesiones en la que los participantes comparten el estado de la canción, el repertorio y hasta el compás exacto que están tocando.

El sistema ayuda al aprendizaje y la enseñanza musical al mostrar cómo realizar los acordes en cada instrumento y al permitir afinarlos.

La arquitectura soporta numerosas vistas complejas y responsivas para distintos instrumentos, y permite agregar nuevas vistas para otros instrumentos. Para las partituras emplea la librería VexFlow, mientras que para las vistas de letra y acordes se usó un desarrollo propio.

El principal desafío técnico fue el de la reproducción en dispositivos distribuidos: un ”delay”de 20 ms empieza a ser perceptible por el oído humano y la latencia en internet es mayor. Se implementó un protocolo que sincroniza los dispositivos usando WebSocket y WebRTC (con compensación de jitter) a través de un servidor Golang.

Para probar y ”debuggear.el sistema de sincronización hubo que desarrollar algunas vistas y controles. Las pruebas de aceptación se desarrollaron de punta a punta con NUnit y Playwright.

Palabras Clave: Web, Vue.js, WebSocket, WebRTC, Diseño responsivo, Golang, NUnit, Playwright, Sincronismo, Compensación de *jitter*, Música, Letras, Acordes, Partituras.

English

Fogón is a distributed system designed to facilitate musical practice and synchronization among musicians. It provides each musician with a progressive web application featuring instrument-specific views: lyrics, chords, or sheet music. It allows users to create playlists and edit songs. Additionally, it enables the creation of sessions where participants share the song state, repertoire, and even the exact bar they are playing.

The system aids in musical learning and teaching by showing how to perform chords on each instrument and allowing for tuning.

The architecture supports numerous complex and responsive views for different instruments, and allows for the addition of new views for other instruments. For sheet music, it employs the VexFlow library, while for lyrics and chord views, a custom development was used.

The main technical challenge was distributed device playback: a delay of 20 ms starts to become perceptible to the human ear, and internet latency is typically higher. A protocol was implemented

that synchronizes devices using WebSocket and WebRTC (with jitter compensation) through a Golang server.

To test and debug the synchronization system, several views and controls had to be developed. End-to-end acceptance tests were developed with NUnit and Playwright.

Keywords: Web, Vue.js, WebSocket, WebRTC, Responsive design, Golang, NUnit, Playwright, Synchronization, Jitter compensation, Music, Lyrics, Chords, Sheet Music.

Agradecimientos

A mi madre y a mi padre,

A la Educación Pública y en particular a las cátedras de Arquitectura de Software, Base de Datos, Introducción a Sistemas Distribuidos y Sistemas Distribuidos de la Universidad de Buenos Aires.

A Pitágoras, a Newton, a Turing y a todos los que asumen la heroica tarea de descubrir y transmitir ciencia.

A los artistas que prefieren estructuras rigurosas.

A las Cadenas de Márkov y la proliferación de IAs.

A vos que estás leyendo esto.

1 Introducción

Tanenbaum y Van Steen definen: “Un sistema distribuido es una colección de computadoras independientes que aparece ante sus usuarios como un sistema único y coherente” [**?**]

La definición coincide con la de un grupo musical que combina armonías, melodías y ritmos de modo que se escuche como un tema único y coherente.

Para hacer esto los músicos se nutren de protocolos y mecanismos de coordinación que les permiten sincronizarse y compartir información.

Los avances en la ciencia y la ingeniería fueron incorporados por los músicos: Pitágoras sintetizó la matemática y la armonía; la imprenta permitió la publicación de partituras; la revolución industrial introdujo el metrónomo de Maelzel.

Desde que existe Internet, circulan archivos con páginas de acordes que evolucionaron a páginas multimedia, archivos MIDI, aplicaciones de edición de partituras, etc.

La aplicación .El Fogón”pone a disposición de los músicos estos avances y propone un nuevo enfoque: cada músico puede acceder con su dispositivo a un Estado compartido de compás, canción, acordes, partituras, repertorio, etc.

Esto permite que varios músicos toquen juntos y en sincronía, compartiendo y actualizando información en tiempo real, pero cada uno ve la información de su instrumento.

1.1 Motivación

Busca hacer un aporte novedoso a la música desde la informática, incorporando soluciones anteriores y agregando un enfoque novedoso: el estado compartido entre músicos.

Además, busca la alta disponibilidad: los músicos pueden acceder a la aplicación en su dispositivo en cualquier momento, sin necesidad de conexión a la red.

1.2 Objetivos

Fogón es una solución dirigida tanto a cantantes y guitarristas aficionados como a músicos de orquestas profesionales: una aplicación en donde puedan buscar letras de acordes y canciones de modo intuitivo y también una herramienta que los ayude a ensayar y crear cosas nuevas.

1.2.1 Objetivo General

Cada músico podrá ver en su dispositivo la vista de su instrumento: el cantante, la letra; el guitarrista, los acordes; el pianista, sus partituras. Tendrá autoscroll y subrayado automático del compás actual; podrá editar los tamaños de letra y acordes.

El público podrá ver y editar una variedad de canciones publicadas en el mismo sitio. Si se loguea con su usuario, podrá compartir sus canciones con otros usuarios.

Varios usuarios podrán unirse en una sesión para sincronizar la lista de canciones, la canción que se está reproduciendo y el compás actual: de este modo podrán organizar un ensayo, un concierto o una

noche de karaoke entre amigos.

1.2.2 Objetivos Técnicos

La sincronización del compás en una sesión debe ser exacta cuando un grupo de músicos está tocando: un delay de 20 ms empieza a ser perceptible por el oído humano y la latencia en Internet puede ser mayor. Los distintos dispositivos se conectarán con un servidor Golang e implementarán un protocolo que combine timestamps sincronizados (basados en NTP), buffers adaptativos y compensación del jitter (variación en la latencia) para resolver esto.

El mismo servidor, además, por medio de HTTP intercambia los archivos de las canciones con las aplicaciones.

Todas las vistas deberán poder adaptarse a distintos dispositivos, ser configurables y extensibles: será posible incorporar vistas adicionales, como notación numérica para armónica, tablaturas para guitarra y reproductores multimedia como YouTube o MIDI.

Edición de letra y acordes: también podrán editar las canciones mediante una interfaz intuitiva y accesible; la compleja relación entre las letras y los acordes, que además se agrupan en partes que se repiten según una secuencia, podrá modificarse de una manera natural y sencilla.

Sincronización: varios usuarios logueados podrán unirse en una sesión para sincronizar la lista de canciones, la canción que se está reproduciendo y el estado de la reproducción.

Construir algunas herramientas necesarias para la música, como un afinador que permita afinar distintos instrumentos.

Construir herramientas para probar y ”debuggear.[el] sistema de sincronización desarrollado.

2 Estado del Arte

Como es parte nuestro ”negocio”, describiremos algunos conceptos sobre la naturaleza del sonido y de la música.

Luego, comentaremos algunas tecnologías destacadas que tomamos en el Fogón y sobre las novedades que llegaron con internet. También, repasaremos aplicaciones web con IA que ofrecen servicios relacionados con la música.

Finalmente, explicaremos cómo nuestro enfoque es novedoso y aporta valor, comparado con las herramientas antes mencionadas.

2.1 Sobre el sonido y la música

.El sonido es una onda mecánica que se propaga a través de un medio elástico, como el aire o el agua.z que sus Característica principales sin la frecuencia, la amplitud y el timbre.

La frecuencia determina el tono o la nota musical, la amplitud el volumen y el timbre la calidad del sonido, permite distinguir entre diferentes instrumentos que tocan la misma nota.

La mayoría de los humanos no podemos determinar la frecuencia a la que vibra una cuerda, pero si podemos distinguir si produce un sonido agradable. Pitágoras descubrió, en el siglo VI a.C.,que para producir sonidos agradables entre dos cuerdas vibrantes tienen que tener relaciones matemáticas. Ej: (2:1)Una octava, (3:2)una quinta justa. (4:3)una cuarta justa.

Todos conocemos la definición de música acerca de hacer algo con la melodia, la armonia y el ritmo. Pues bien, la melodia es como cambia la frecuencia en el tiempo, la armonia es la combinacion de varias frecuencias. El ritmo es la repetición de sonidos (o volumenes o intenciones) en intervalos regulares. En la música suelen organizarse de a 2, 3 o 4 tiempos, formando compases.

2.2 Herramientas

2.2.1 Pentagramas

15 Siglos antes que Pitágoras, por el 2000 A.C., en la Mesopotamia se usaban tablillas de arcilla para anotar música, con simbolos que representaban las alturas de las notas.

Recien en el siglo IX, el monje benedictino Guido d’Arezzo desarrollo un sistema de notacion musical basado en cuatro lineas y espacios, que luego evolucionó al pentagrama con cinco lineas que usamos hoy.

solo 50 años años despues de la invencion de la imprenta, Ottaviano Petrucci, en 1501, publica ”Harmonice Musices Odhecaton”, con las primeras partituras impresas con tipos móviles, permitiendo la difusion masiva de la musica escrita.

2.2.2 Cancioneros

En el siglo XIX, se transmitian los acordes folcloricos de forma oral, pero para el siglo XX, en la musica popular (tango, rock, jazz) se empieza a usar la notacion de acordes sobre la letra de la cancion.

Se difunden las revistas de acordes y letras, y luego los cancioneros impresos. Con la llegada de internet, surgen diversas paginas y aplicaciones que revisamos en la proxima seccion.

2.2.3 Afinadores

En el siglo XVII, Marin Mersenne formuló las leyes que gobiernan la vibración de cuerdas tensas, permitiendo una base física para afinar cuerdas y abrió el camino hacia la acústica moderna.

$$f = \frac{1}{2L}\sqrt{\frac{T}{\mu}}$$

donde f es la frecuencia, L es la longitud de la cuerda, T es la tensión aplicada y μ es la densidad lineal de masa.

Se afinaba con un diapasón, un metal que siempre sonaba con la misma frecuencia y que se tomaba como nota de referencia.

En 1939 durante la conferencia de Londres científicos y músicos acordaron que la nota La4 se afinaría a 440 Hz, es decir, 440 vibraciones por segundo.

En 1980 se popularizaron afinadores digitales compactos, portátiles y precisos.

2.2.4 Metrónomos

En 1815 Johann Maelzel patenta el metrónomo mecánico, ganándose el reconocimiento de Beethoven y popularizándose en toda Europa.

En el siglo XX llegaron metrónomos electrónicos, que permiten mayor precisión y funcionalidades adicionales como sonidos personalizables, luces intermitentes y conectividad con otros dispositivos.

2.3 Aplicaciones Web Musicales

Con dispositivos moviles inteligentes, escuchar un sonido y procesarlo para detectar su frecuencia es muy sencillo, igual que repetir un comportamiento cada un periodo constante de tiempo. Es por eso, que hubo metronomos y afinadores en cada celular desde el comienzo de este siglo.

Las proximas lineas transcurriran sobre aplicaciones web que son usadas actualmente por musicos, pero ademas de señalar solamente sus aportes, se hara foco en sus debilidades.

2.3.1 Cancioneros

los cancioneros online Reemplazaron a los folletines ofreciendo la diversidad de la web pero copiaron un defecto.

El musico esta tocando su instrumento, ajustado perfecto con el metronomo, cuando llega al ultimo acorde de la pagina (o de la pantalla) y lo obliga a soltar su intrumento para manipular el cancionero, perdiendo asi el ritmo.

Las paginas suelen en su mayoria estar diagramadas en la pantalla pensado en maximizar el espacio para publicidad y no en la usabilidad del musico, por lo que ofrecen poca personalizacion en la vista. Esto hace que aunque muchas ofrezcan instrucciones sobre como se realizan los acordes en el instrumento, o algunos implementen un rudimentario autoscroll, sea poco visible para el musico.

De las que tienen contenido argentino, las principales son lacuerda.net, cifraclub y acordesweb.

Y claro, muestra solo los acordes, para las paginas de acordes son paginas distintas!

2.3.2 Reproductores online de video y acordes

Otras aplicaciones web ofrecen videos con acordes sincronizados, como Chordify y Ultimate Guitar, ajustan la reproduccion a videos de YouTube, apenas permite editar los acordes

2.3.3 Editores de partituras online

Los editores de partituras online permiten crear, editar y compartir partituras musicales a través de una interfaz web. musescore.com destaca por su buen balance entre contenido gratuito y de pago y su comunidad activa de usuarios,

2.3.4 Reproductores online de audio y video

Como YouTube y Spotify, permiten reproducir audio y video en línea, pero no están diseñados para músicos que tocan juntos. Estas aplicaciones suelen permitir compartir una lista de reproducción.

2.4 El futuro llego, hace rato...

Al momento de escribir esto, diciembre de 2025, los resultados del uso de la IA vienen avanzado enorme y exponencialmente.

Logran generar todo tipo de contenido, letras y partituras, audios y videos.

Tambien permiten procesar audio para separar los instrumentos y transcribir partituras. Sobresale en ese sentido la aplicacion Moises.ai y Spleeter.

2.5 Conclusión

Mostramos una tabla comparativa de las herramientas presentadas:

Característica	Metrónomo	Diapasón	Cancioneros	Cancioneros Web	Reprod. video y acordes	Edit. partituras online	Reprod. audio y video	Herramientas Con IA	Fogon.ar
Marca el ritmo?	Sí	No	No	No	No	No	No	Sí	Sí
Afina?	No	Sí	No	No	No	No	No	Sí	Sí
Muestra acordes?	No	No	Sí	Sí	Sí	Sí	No	Sí	Sí
Se actualiza con la canción?	No	No	No	Algunos	Sí	No	No	Sí	Sí
Muestra partitura?	No	No	No	No	No	Sí	No	Sí	Sí
Funciona sin Internet?	Sí	Sí	Sí	No	No	No	No	No	Sí
Permite estados compartidos?	No	No	No	No	No	No	Algunos	No	Sí

Cuadro 1: Comparativa de herramientas y aplicaciones musicales

Se deduce entonces que el estado compartido es un enfoque novedoso para aplicaciones web dedicadas a musicos.

La gran cantidad de tipos de aplicaciones web musicales existentes, cada una con su solucion particular, muestra la necesidad de unificar herramientas para mejorar la experiencia del musico.

El criterio de privilegar el aspecto educativo y de usabilidad esta ausente en los cancioneros online y en la mayoria de las aplicaciones web musicales.

3 Solución Implementada

En esta seccion esta la descripcion de la solución implementada y una explicacion tecnica basada en el modelo de vistas 4+1 de Kruchten [**?**].

En la última vista, la vista de escenarios, los escribimos de igual modo que a las pruebas de aceptación en Reqnroll [**?**].

Por último, incluimos una revision de las tecnologías utilizadas.

3.1 WWW.FOGON.AR

Es una aplicación progresiva (PWA) y multiplataforma que permite buscar y tocar canciones, y tambien crear, editar y compartirlas.

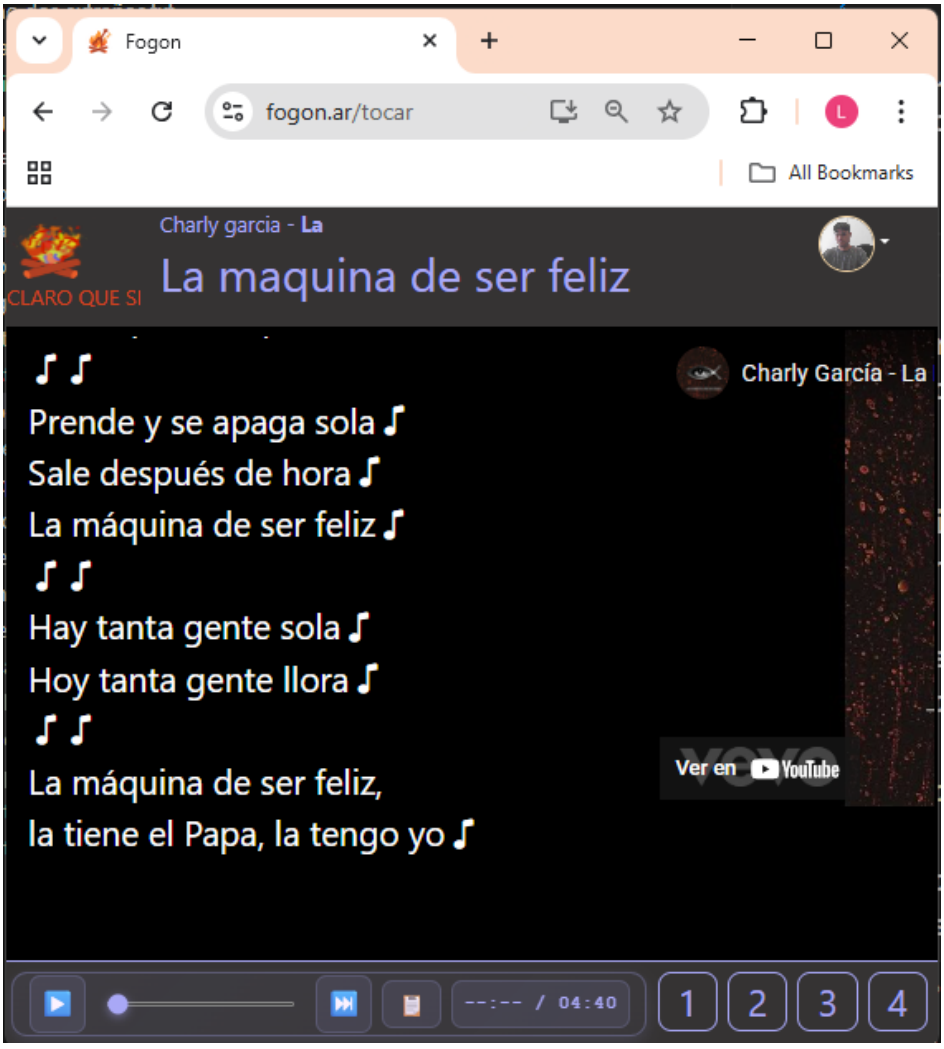

Figura 1: FIG 1: ”La maquina de ser feliz”, tocando con un video

Ademas, Permite crear sesiones colaborativas para llevar , en un estado compartido , el compas y la cancion, de modo que cada musico vea las instruccones para su instrumento, en su dispositivo, de manera sincronizada.

Administra la lista de reproducción y listas en general, guardandolas localmente o en el servidor.
Ofrece tambien otras herramientas utiles para los musicos, como un afinador y cambios automáticos de escala.

3.1.1 tocar

La pagina tocar muestra la vista segun el musico:
[ACA, FOTO DE VARIAOS EXPLORADORES MOSTRANODO LA PAGINA TOACAR CON DISTINTOS INSTRUMENTOS]
En la cabecera, muestra los datos de la cancion y permite cambiar la vista y la Configuración.
A lo largo de la pantalla, la vista particular del instrumento.
Debajo, el control de reproduccion y el metrónomo.

icono-fogon Arriba a la izquierda, el icono del fogon muestra el estado general de la aplicación. Marca el pulso cuando estan tocando y muestra los usuarios en la sesión.
[ACA, FOTO DEL ICONO FOGON]

menu Arriba a la derecha, el icono de menu abre las opciones para configurar la vista y el usuario.

vistas Para configurar su vista, cada musico accede desde el menu a ”Ver”:
[ACA, FOTO DEL MENU DESPLEGADO VER]
Desde aqui puede elegir entre ver letras y/o acordes o la partitura. Ademas, puede ajustar el tamaño de la letra, los acordes y la partitura para ajustarla a cada dispositivo.
En la opcion de reproduccion, puede elegir acompañar la reproduccion con un video o con un midi generado a travez de las partituras.
En las ultimas opciones, se ajuste la cantidad de columnas que se muestra en pantalla y si muestran instrucciones para el musico, la secuencia de acordes o la pantalla para reproducir MIDIs.

3.1.2 Sincronizar

Descripción de la funcionalidad de sincronización.
Cuando un usuario crea un fogon, puede invitar a otros a unirse a la sesion colaborativa. Desde el menú, tambien, puede asignar distintos roles a cada usuario:
[FOTO DE ASIGNAR ROLES]
Todos los usuarios ven la misma cancion y el mismo compas, pero cada uno ve las instrucciones para su instrumento, en su dispositivo. Los administradores pueden cambiar la cancion y controlar la reproduccion. El director, es el único capaz de reproducir algun video o audio.

3.1.3 Buscar y Listar

El fogon, ademas de permitir buscar canciones por banda o por artista, tienen una serie de filtros multiopcion

[FOTO BUSQUEDA]

De este modo, permite buscar por:

* Escala * Etiquetas * Calidad * Tempo * Si tiene o no Videos o Partituras * Cantidad de Acordes * Cantidad de Partes * Duracion

Permite armar listas de reproduccion tanto locales como en el servidor.

3.1.4 Editar

En la pantalla de Configuración se administran los datos basicos del usuario.

[ACA, FOTO DEL INICIO DE CONFIGURACION]

Desde aqui el musico puede cambiar su nombre, el instrumento que toca y la imagen que representa su dispositivo.

Configura de que modo se crean sus sesiones: el nombre y el rol default de los nuevos usuarios que se unan a la sesion.

Permite configurar el usuario para conectarse con el servidor.

Ademas, en esta pantalla, desde la barra superior, se puede acceder a las secciones: ”Hola”, .Afinar”, .Acerca de...”

Hola

Acerca de...

Afinar

3.1.5 Configurar

En la pantalla de Configuración se administran los datos basicos del usuario.

[ACA, FOTO DEL INICIO DE CONFIGURACION]

Desde aqui el musico puede cambiar su nombre, el instrumento que toca y la imagen que representa su dispositivo.

Configura de que modo se crean sus sesiones: el nombre y el rol default de los nuevos usuarios que se unan a la sesion.

Permite configurar el usuario para conectarse con el servidor.

Ademas, en esta pantalla, desde la barra superior, se puede acceder a las secciones: ”Hola”, .Afinar”, .Acerca de...”

3.2 Arquitectura del Sistema

Para describir la arquitectura del sistema, utilizamos el modelo de vistas 4+1 de Kruchten [**?**].
En la vista lógica detallaremos componentes principales del cliente y las clases que permiten comunicarse con el servidor. Luego, en la vista de procesos, veremos el protocolo para sincronización y mantener el estado compartido entre sesiones. En la vista física veremos el mapeo físico en la red en la que ocurren dichos procesos. El 4 del 4+1 es la vista de desarrollo, donde describimos la organizacion del codigo. Y el +1: la vista de escenarios, que seran descriptos en formato Reqnroll. [**?**].

3.2.1 Vista Lógica

La vista lógica describe la funcionalidad que el sistema ofrece a sus usuarios finales, organizando el sistema en componentes lógicos según sus responsabilidades.

Componentes Principales El sistema El Fogón se organiza en los siguientes componentes lógicos principales:

- **Gestor de Canciones:** Administra el repositorio de canciones, incluyendo búsqueda, listado, creación y edición. Gestiona la estructura de letra, acordes, partituras y secuencias de partes.
- **Motor de Renderizado:** Responsable de generar las vistas específicas para cada instrumento (letra, acordes, partituras). Implementa el autoscroll y el resaltado del compás actual. Utiliza VexFlow para la notación musical y un motor propio para letra y acordes.
- **Motor de Sincronización:** Coordina el estado compartido entre múltiples dispositivos en una sesión. Gestiona el compás actual, la canción activa y el repertorio. Implementa compensación de latencia y jitter.
- **Gestor de Sesiones:** Administra las sesiones colaborativas, permitiendo que múltiples usuarios se unan y compartan estado. Controla roles y permisos de los participantes.
- **Motor de Reproducción:** Controla la reproducción de canciones con metrónomo, videos sincronizados (YouTube) y MIDI generado desde partituras.
- **Herramientas Musicales:** Provee funcionalidades auxiliares como afinador, transposición automática de tonalidad y visualización de diagramas de acordes para diferentes instrumentos.
- **Gestor de Configuración:** Administra las preferencias del usuario: instrumento, tamaño de fuentes, columnas, modo de reproducción, y configuración de sesiones.
- **Gestor de Persistencia:** Maneja el almacenamiento local (IndexedDB/LocalStorage) y la sincronización con el servidor remoto. Permite operación offline.

- **Gestor de Usuarios:** Administra autenticación, perfiles de usuario y compartición de canciones entre usuarios.

Relaciones entre Componentes

- El **Motor de Renderizado** consulta al **Gestor de Canciones** para obtener el contenido y al **Gestor de Configuración** para las preferencias de visualización.
- El **Motor de Sincronización** notifica al **Motor de Reproducción** sobre cambios en el compás y éste actualiza el **Motor de Renderizado**.
- El **Gestor de Sesiones** coordina con el **Motor de Sincronización** para mantener el estado compartido entre dispositivos.
- El **Gestor de Persistencia** provee datos tanto al **Gestor de Canciones** como al **Gestor de Usuarios**.

3.2.2 Vista de procesos

La vista de procesos describe los aspectos dinámicos del sistema, incluyendo los procesos concurrentes, la comunicación entre ellos, y cómo el sistema maneja la sincronización y el rendimiento.

Procesos Principales

- **Proceso de Renderizado de UI:** Thread principal de la aplicación Vue.js que maneja el ciclo de vida de los componentes y la actualización del DOM.
- **Worker de Sincronización:** Maneja la conexión WebSocket con el servidor y procesa los mensajes de sincronización en tiempo real sin bloquear la UI.
- **Worker de Audio:** Procesa señales de audio para el afinador y genera eventos de audio para el metrónomo con precisión temporal.
- **Proceso Servidor (Golang):** Múltiples goroutines que manejan conexiones WebSocket, HTTP, y la lógica de distribución de estado.

Protocolo de Sincronización El desafío principal es mantener sincronizados los dispositivos cuando la latencia de red puede superar los 20ms (umbral de percepción humana).

Mecanismo implementado:

1. **Sincronización de Relojes:** Al unirse a una sesión, los clientes sincronizan sus relojes con el servidor usando un protocolo simplificado similar a NTP:

   - Cliente envía timestamp t_1
   - Servidor responde con t_1, su timestamp t_s y timestamp de respuesta t_2
   - Cliente calcula offset: $offset = \frac{(t_2 - t_1) - (t_4 - t_3)}{2}$
   - Se repite múltiples veces y se toma la mediana

2. **Broadcast de Eventos:** Cuando un usuario cambia el compás:
   - Genera evento con timestamp sincronizado futuro ($t_{play} = t_{now} + \Delta$)
   - Servidor broadcast a todos los clientes
   - Cada cliente programa el cambio para t_{play}
3. **Buffer Adaptativo:** Δ se ajusta dinámicamente según la latencia medida de la sesión.
4. **Compensación de Jitter:** Se implementa un buffer de reproducción que absorbe variaciones en la latencia, manteniendo la periodicidad del metrónomo.

Comunicación entre Procesos

- **WebSocket:** Canal bidireccional para sincronización en tiempo real del estado compartido (compás, canción, lista).
- **WebRTC:** Propuesto para comunicación peer-to-peer en caso de falla del servidor o para reducir latencia en redes locales.
- **HTTP/REST:** Para operaciones no críticas en tiempo: búsqueda, carga/guardado de canciones, autenticación.
- **Message Bus interno:** Patrón Observer para comunicación entre componentes Vue.js.

Gestión de Concurrencia

- **En el Cliente:** JavaScript es single-threaded, pero usa Web Workers para procesamiento pesado (audio, sincronización) sin bloquear la UI.
- **En el Servidor:** Golang maneja concurrencia con goroutines y channels. Cada sesión tiene una goroutine dedicada que serializa las actualizaciones de estado, evitando race conditions.
- **Resolución de Conflictos:** En caso de eventos simultáneos, el servidor actúa como autoridad y resuelve mediante timestamps (el evento con timestamp menor gana).
- Componente 1

- Componente 2
- Componente 3

3.2.3 Vista Física

La vista física describe cómo los componentes de software se mapean en la infraestructura de hardware y la topología de red del sistema.

Arquitectura de Despliegue El sistema El Fogón utiliza una arquitectura cliente-servidor con capacidad de operación offline:

- **Nodos Cliente:** Dispositivos heterogéneos (smartphones, tablets, laptops, desktops) que ejecutan la PWA en navegadores web modernos.
- **Servidor Central:** Servidor Golang que hospeda:
  - Servidor HTTP/HTTPS para servir la aplicación web
  - Servidor WebSocket para sincronización en tiempo real
  - API REST para operaciones CRUD
  - Base de datos de canciones y usuarios
- **CDN (Opcional):** Para distribución de assets estáticos (imágenes, CSS, JS).

Topología de Red **Topología Estrella:** Todos los clientes se conectan al servidor central. El servidor actúa como hub de comunicación para las sesiones sincronizadas.

Ventajas:

- Simplicidad en la gestión de estado compartido
- El servidor es la única fuente de verdad
- Facilita la resolución de conflictos

Desventajas:

- Punto único de falla
- Latencia agregada (cliente A $\rightarrow$ servidor $\rightarrow$ cliente B)

Mitigación con WebRTC: En desarrollo - topología híbrida donde los clientes pueden conectarse peer-to-peer para reducir latencia en LANs, manteniendo el servidor como fallback.

Distribución de Responsabilidades

- **En el Cliente:**
  - Renderizado de vistas (Vue.js + VexFlow)
  - Lógica de presentación y UI
  - Cache local de canciones (IndexedDB)
  - Service Worker para operación offline
  - Compensación de jitter y scheduling preciso de eventos
- **En el Servidor:**
  - Autoridad sobre el estado compartido
  - Persistencia de canciones y usuarios (base de datos)
  - Broadcast de eventos de sincronización
  - Autenticación y autorización
  - Mediación de timestamps para sincronización

Requisitos de Hardware y Red

- **Cliente:** Navegador moderno (Chrome 90+, Firefox 88+, Safari 14+), 2GB RAM mínimo, conexión a internet opcional.
- **Servidor:** CPU de 2+ cores, 4GB RAM, 20GB almacenamiento, conexión de banda ancha (10 Mbps mínimo).
- **Red:** Para sincronización en tiempo real, se recomienda latencia ¡100ms y conexión estable. El sistema adapta el buffer según latencia detectada.

Escalabilidad El servidor Golang puede escalar horizontalmente con:

- Balanceador de carga para HTTP/HTTPS
- Sticky sessions para WebSocket
- Redis para compartir estado entre instancias del servidor

cómo se despliega en hardware (nodos, servidores, redes).

- Componente 1
- Componente 2
- Componente 3

3.2.4 Vista de Desarrollo

La vista de desarrollo describe la organización del software desde la perspectiva del programador, incluyendo la estructura de módulos, gestión de código fuente y estrategia de build.

Organización del Código El proyecto se organiza en dos repositorios principales:

Repositorio Cliente (fogon-web):

- `/src/components/` - Componentes Vue.js reutilizables
  - `/views/` - Vistas de instrumentos (LetraView, AcordesView, PartituraView)
  - `/controls/` - Controles de reproducción y configuración
  - `/tools/` - Herramientas (Afinador, Metrónomo)
- `/src/services/` - Lógica de negocio e integración
  - `song-service.ts` - Gestión de canciones
  - `sync-service.ts` - Protocolo de sincronización
  - `session-service.ts` - Gestión de sesiones
  - `api-client.ts` - Cliente HTTP para el backend
- `/src/stores/` - Estado global (Pinia/Vuex)
- `/src/workers/` - Web Workers para procesamiento
- `/src/models/` - Tipos y modelos de datos (TypeScript)
- `/src/utils/` - Utilidades compartidas
- `/public/` - Assets estáticos

Repositorio Servidor (fogon-server):

- `/cmd/server/` - Punto de entrada de la aplicación
- `/internal/handlers/` - Handlers HTTP y WebSocket
- `/internal/services/` - Lógica de negocio
- `/internal/models/` - Estructuras de datos
- `/internal/repository/` - Capa de persistencia
- `/internal/sync/` - Protocolo de sincronización
- `/pkg/` - Paquetes reutilizables

Stack Tecnológico Frontend:

- **Framework:** Vue.js 3 - Composition API para componentes reactivos
- **Lenguaje:** TypeScript - Tipado estático para mayor robustez
- **Build Tool:** Vite - Build rápido y HMR
- **Estado:** Pinia - State management
- **Routing:** Vue Router - Navegación SPA
- **Notación Musical:** VexFlow - Renderizado de partituras
- **PWA:** Workbox - Service Worker y cache
- **Estilos:** CSS3 + SCSS - Diseño responsivo

Backend:

- **Lenguaje:** Go 1.21+ - Concurrencia nativa con goroutines
- **WebSocket:** gorilla/websocket - Comunicación bidireccional
- **HTTP Router:** chi o gorilla/mux
- **Base de Datos:** PostgreSQL para datos estructurados, Redis para cache/sessions
- **ORM:** GORM o sqlx

Testing:

- **Unit Tests Frontend:** Vitest + Vue Test Utils
- **Unit Tests Backend:** Go testing package + testify
- **E2E Tests:** Playwright + .NET (C#) + Reqnroll (BDD)
- **API Tests:** Postman/Newman

DevOps:

- **Control de Versiones:** Git + GitHub
- **CI/CD:** GitHub Actions
- **Containerización:** Docker + Docker Compose
- **Deployment:** (TBD - cloud provider)

Gestión de Dependencias

- **Frontend:** npm/pnpm con `package.json` y lockfile
- **Backend:** Go modules (`go.mod`, `go.sum`)

Estrategia de Build

- **Desarrollo:** Vite dev server con HMR para frontend, `go run` con hot-reload para backend
- **Testing:** Pipeline de CI ejecuta tests unitarios y E2E en cada PR
- **Producción:** Build optimizado con minificación, tree-shaking, y code-splitting. Docker multi-stage build para deployments eficientes.

Patrones de Diseño Utilizados

- **Observer:** Para reactivity en Vue.js y eventos de sincronización
- **Factory:** Para creación de componentes de vista según instrumento
- **Strategy:** Para diferentes motores de renderizado (letra, acordes, partitura)
- **Repository:** Para abstracción de persistencia en el backend
- **Singleton:** Para servicios globales (conexión WebSocket, configuración)

3.2.5 Vista de Escenarios

La vista de escenarios (la -1"del modelo de Kruchten) ilustra cómo las otras cuatro vistas trabajan juntas a través de casos de uso concretos. Documentamos los escenarios principales usando el formato de pruebas de aceptación de Reqnroll (Gherkin).

Escenario 1: Tocar una canción en solitario **Contexto:** Un músico quiere buscar y tocar una canción desde su dispositivo.

Listing 1: Escenario: Búsqueda y reproducción de canción

```
Feature: Tocar canciones
  Como musico
  Quiero buscar y tocar canciones
  Para practicar con mi instrumento

Scenario: Buscar y reproducir una cancion
  Given que soy un usuario no autenticado
```

```
  When ingreso "la maquina de ser feliz" en el buscador
  Then veo una lista de resultados con esa cancion
  When selecciono "La maquina de ser feliz - Charly Garcia"
  Then veo la vista de mi instrumento (acordes para guitarra)
  When presiono el boton de reproducir
  Then comienza el autoscroll sincronizado con el metronomo
  And el compas actual se resalta automaticamente
```

Vistas involucradas:

- **Lógica:** Gestor de Canciones, Motor de Renderizado, Motor de Reproducción
- **Procesos:** Proceso de renderizado UI, scheduling de eventos del metrónomo
- **Física:** Ejecución local en el cliente, sin comunicación con servidor
- **Desarrollo:** Componentes Vue (SearchView, PlayerView), Services (SongService, PlaybackService)

Escenario 2: Crear sesión sincronizada **Contexto:** Un grupo de músicos quiere ensayar tocando cada uno desde su dispositivo pero sincronizados.

Listing 2: Escenario: Sesión colaborativa sincronizada

```
Feature: Sesiones sincronizadas
  Como musico lider de una banda
  Quiero crear una sesion compartida
  Para que todos toquemos sincronizados

Scenario: Crear y unirse a una sesion
  Given que soy un usuario autenticado "Luis (Guitarra)"
  When creo una nueva sesion "Ensayo Banda"
  Then obtengo un codigo de sesion "ABC123"

  Given que otro usuario "Ana (Voz)" ingresa el codigo "ABC123"
  When Ana se une a la sesion
  Then veo que Ana esta conectada en mi sesion
  And Ana ve el mismo estado de la sesion que yo

  When selecciono la cancion "Flaca - Andres Calamaro"
  Then Ana ve automaticamente la misma cancion
  And Ana ve la vista de su instrumento (letra)
```

```
  And yo veo la vista de mi instrumento (acordes)

  When inicio la reproduccion
  Then todos los dispositivos comienzan sincronizados
  And el compas se actualiza simultaneamente en todos
  And la diferencia de tiempo es menor a 20ms
```

Vistas involucradas:

- **Lógica:** Gestor de Sesiones, Motor de Sincronización, Gestor de Usuarios
- **Procesos:** Protocolo de sincronización de relojes, WebSocket bidireccional, compensación de jitter
- **Física:** Clientes múltiples, servidor central coordinando, topología estrella
- **Desarrollo:** SessionService, SyncService, WebSocket handlers en Go

Escenario 3: Editar y compartir canción **Contexto:** Un usuario quiere corregir los acordes de una canción y compartirla con la comunidad.

Listing 3: Escenario: Edición colaborativa

```
Feature: Edicion de canciones
  Como musico
  Quiero editar canciones y compartirlas
  Para contribuir a la comunidad

Scenario: Editar y publicar una cancion
  Given que estoy tocando "Como dos extranos - Los Fabulosos"
  When presiono el boton "Editar"
  Then ingreso al modo de edicion
  When cambio el acorde del compas 5 de "Am" a "Am7"
  And agrego una nueva parte "Puente" con su letra y acordes
  And modifico la secuencia para incluir el puente
  Then veo una vista previa de mis cambios

  When guardo la cancion
  Then la cancion se guarda localmente
  And tengo la opcion de "Publicar al servidor"

  When publico la cancion
```

```
  Then otros usuarios pueden encontrarla en sus busquedas
  And la cancion aparece en mi perfil como contribucion
```

Vistas involucradas:

- **Lógica:** Gestor de Canciones (editor), Gestor de Persistencia, Gestor de Usuarios
- **Procesos:** Validación de formato, serialización/deserialización, API REST
- **Física:** Guardado local primero (IndexedDB), luego sincronización con servidor
- **Desarrollo:** EditorView component, SongEditor service, API endpoints en Go

Escenario 4: Uso offline **Contexto:** Un músico quiere practicar en un lugar sin internet.

Listing 4: Escenario: Operación offline

```
Feature: Modo offline
  Como musico
  Quiero usar la aplicacion sin internet
  Para practicar en cualquier lugar

Scenario: Tocar canciones previamente cargadas sin conexion
  Given que previamente cargue 10 canciones con conexion
  And marque las canciones como "favoritas"
  When pierdo la conexion a internet
  And abro la aplicacion
  Then la aplicacion carga normalmente (PWA)
  And puedo acceder a mis canciones favoritas
  And puedo tocar con autoscroll y metronomo
  But no puedo crear sesiones sincronizadas
  And no puedo buscar canciones nuevas

  When edito una cancion en modo offline
  Then los cambios se guardan localmente
  And veo una indicacion "Pendiente de sincronizar"

  When recupero la conexion a internet
  Then mis cambios se sincronizan automaticamente con el servidor
```

Vistas involucradas:

- **Lógica:** Gestor de Persistencia (prioridad local), Service Worker

- **Procesos:** Cache-first strategy, background sync cuando se recupera conexión
- **Física:** Ejecución 100 % local en el cliente
- **Desarrollo:** Service Worker (Workbox), IndexedDB para persistencia, sync queues

Cobertura de Pruebas Estos escenarios se implementan como pruebas de aceptación automatizadas usando:

- **Framework:** Reqnroll (.NET/C#) para BDD
- **Automatización:** Playwright para control del navegador
- **Ejecución:** CI/CD pipeline ejecuta estos tests en cada merge a main

Los escenarios validan la integración completa de las cuatro vistas arquitectónicas y garantizan que el sistema cumple con los requisitos funcionales desde la perspectiva del usuario final.

3.3 Tecnologías Utilizadas

Listar y justificar las tecnologías seleccionadas.

4 Metodología Aplicada

Describir la metodología empleada para el desarrollo del trabajo.

4.1 Marco Metodológico

Explicar el marco metodológico utilizado (ágil, cascada, etc.).

4.2 Proceso de Desarrollo

Detallar las etapas del proceso de desarrollo:

1. Análisis de requisitos
2. Diseño de la solución
3. Implementación
4. Pruebas
5. Despliegue

4.3 Herramientas de Desarrollo

- Herramienta 1
- Herramienta 2
- Herramienta 3

4.4 Gestión del Proyecto

Describir cómo se gestionó el proyecto (sprints, reuniones, etc.).

5 Experimentación y Validación

Presentar los experimentos realizados y la validación de la solución implementada.

5.1 Diseño de Experimentos

Describir el diseño de los experimentos realizados para validar la solución.

5.2 Casos de Prueba

Detallar los casos de prueba ejecutados.

5.3 Resultados Obtenidos

Presentar los resultados obtenidos del trabajo realizado.

Caso de Prueba	Resultado Esperado	Resultado Obtenido
Prueba 1	Éxito	Éxito
Prueba 2	Éxito	Éxito

Cuadro 2: Resultados de las pruebas realizadas

5.4 Análisis de Resultados

Analizar e interpretar los resultados obtenidos.

5.5 Validación con Usuarios

Describir el proceso de validación con usuarios finales.

6 Cronogramas

Presentar el cronograma de actividades del proyecto.

6.1 Cronograma Planificado

Descripción del cronograma inicial planificado para el proyecto.

Actividad	Duración	Período
Análisis de requisitos	2 semanas	Enero
Diseño	3 semanas	Febrero
Implementación	8 semanas	Marzo-Abril
Pruebas	2 semanas	Mayo
Documentación	1 semana	Mayo

Cuadro 3: Cronograma planificado del proyecto

6.2 Cronograma Real

Descripción del cronograma real de ejecución del proyecto.

6.3 Desviaciones y Ajustes

Análisis de las desviaciones respecto al plan original y los ajustes realizados.

7 Riesgos y Lecciones Aprendidas

Análisis de los riesgos identificados durante el proyecto y las lecciones aprendidas.

7.1 Identificación de Riesgos

Listar y describir los riesgos identificados al inicio del proyecto.

Riesgo	**Probabilidad**	**Impacto**
Riesgo técnico 1	Alta	Alto
Riesgo de recursos	Media	Medio
Riesgo de tiempo	Baja	Alto

Cuadro 4: Matriz de riesgos del proyecto

7.2 Gestión de Riesgos

Describir cómo se gestionaron los riesgos durante el proyecto.

7.3 Problemas Encontrados

Detallar los problemas principales enfrentados durante el desarrollo.

7.4 Lecciones Aprendidas

Compartir las lecciones aprendidas durante la ejecución del proyecto:

- Lección 1
- Lección 2
- Lección 3

8 Impactos Sociales y Ambientales del Proyecto

Análisis de los impactos sociales y ambientales del proyecto desarrollado.

8.1 Impacto Social

Describir el impacto social esperado o generado por el proyecto:

- Beneficios para la comunidad
- Mejoras en la calidad de vida
- Accesibilidad y inclusión

8.2 Impacto Ambiental

Analizar el impacto ambiental del proyecto:

- Consumo de recursos
- Huella de carbono
- Sostenibilidad de la solución

8.3 Responsabilidad Social y Ética

Consideraciones éticas y de responsabilidad social del proyecto.

8.4 Medidas de Mitigación

Describir las medidas implementadas para minimizar impactos negativos.

9 Desarrollos Futuros

Describir posibles extensiones, mejoras y trabajos futuros relacionados con el proyecto.

9.1 Mejoras Propuestas

Listar las mejoras que podrían implementarse en el futuro:

- Mejora 1
- Mejora 2
- Mejora 3

9.2 Funcionalidades Adicionales

Describir funcionalidades adicionales que podrían agregarse.

9.3 Escalabilidad

Analizar cómo el sistema podría escalarse para soportar mayor carga o alcance.

9.4 Investigación Futura

Proponer líneas de investigación futuras relacionadas con el proyecto.

9.5 Conclusiones Finales

Presentar las conclusiones finales del trabajo, resumiendo los principales hallazgos y logros alcanzados.

Referencias

Referencias

[1] A. S. Tanenbaum and M. Van Steen, *Distributed Systems: Principles and Paradigms*, 3rd ed. Boston, MA: Pearson, 2017.

[2] P. Kruchten, ”The 4+1 view model of architecture,”*IEEE Software*, vol. 12, no. 6, pp. 42-50, Nov. 1995.

[3] Reqnroll: Open-source Cucumber-style BDD test automation framework for .NET,Reqnroll. [Online]. Available: https://reqnroll.net/. [Accessed: Jan. 7, 2026].

[4] ”Moises AI Studio: Lyric Writer,”Moises AI Studio. [Online]. Available: https://studio.moises.ai/lyric-writer/. [Accessed: Jan. 7, 2026].