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 **3**

Abstracto **4**
Español . . . 4
English . . . 4

Agradecimientos **6**

1. Introducción **7**
1.1. Motivación . . . 7
1.2. Objetivos . . . 7
1.2.1. Objetivo General . . . 7
1.2.2. Objetivos Técnicos . . . 8

2. Estado del Arte **9**
2.1. Sobre el sonido y la música . . . 9
2.2. Herramientas . . . 9
2.2.1. Pentagramas . . . 9
2.2.2. Cancioneros . . . 10
2.2.3. Afinadores . . . 10
2.2.4. Metrónomos . . . 10
2.3. Aplicaciones Web Musicales . . . 10
2.3.1. Cancioneros . . . 10
2.3.2. Reproductores online de video y acordes . . . 11
2.3.3. Editores de partituras online . . . 11
2.3.4. Reproductores online de audio y video . . . 11
2.4. El futuro llegó, hace rato... . . . 11
2.5. Conclusiones . . . 11

3. Solución Implementada **13**
3.1. WWW.FOGON.AR . . . 13
3.1.1. Tocar . . . 14
3.1.2. Sincronizar . . . 15
3.1.3. Buscar y Listar . . . 16
3.1.4. Editar . . . 17
3.1.5. Configurar . . . 19
3.2. Arquitectura del Sistema . . . 20
3.2.1. Vista Lógica . . . 20
3.2.2. Vista de procesos . . . 22
3.2.3. Vista Física . . . 24
3.2.4. Vista de Desarrollo . . . 25
3.2.5. Vista de Escenarios . . . 27
3.3. Tecnologías Utilizadas . . . 28
3.3.1. Frontend . . . 28
3.3.2. Backend . . . 29
3.3.3. Comunicación y Sincronización . . . 29
3.3.4. Base de Datos . . . 29
3.3.5. Pruebas y Calidad de Código . . . 29
3.3.6. Gestión y Despliegue (DevOps) . . . 29
3.3.7. Otras Herramientas . . . 30

4. Metodología Aplicada **31**
4.1. Metodología del trabajo . . . 31
4.2. Plan . . . 31
4.2.1. Tocar . . . 31
4.2.2. Sincronizar . . . 31
4.2.3. Editar . . . 32

4.2.4. Buscar y listar . . . 33
4.2.5. Tocar y editar partituras . . . 33
4.2.6. Más instrumentos y detalles . . . 34
4.3. Herramientas Utilizadas . . . 34
4.4. Desarrollo de la aplicación . . . 34

5. Experimentación y Validación **36**
5.1. Calibración . . . 36
5.2. Pruebas Unitarias . . . 41
5.3. Pruebas Aceptación . . . 42

6. Conclusiones **43**
6.1. Desarrollos Futuros . . . 43
6.2. Lecciones aprendidas . . . 43
6.3. Conclusiones . . . 44

Referencias **45**

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.

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.

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” [1]

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."

y que sus características principales son 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 melodía, la armonía y el ritmo. Pues bien, la melodía es cómo cambia la frecuencia en el tiempo, la armonía es la combinación de varias frecuencias. El ritmo es la repetición de sonidos (o volúmenes 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

decíamos que Pitágoras descubrió que para producir sonidos agradables entre dos cuerdas vibrantes tienen que tener relaciones matemáticas. Pero ya 15 siglos, por el 2000 A.C., en la Mesopotamia, se usaban tablillas de arcilla para anotar música, con símbolos que representaban las alturas de las notas. Recién en el siglo IX, el monje benedictino Guido d'Arezzo desarrolló un sistema de notación musical basado en cuatro líneas y espacios, que luego evolucionó al pentagrama con cinco líneas que usamos hoy.

Solo 50 años después de la invención de la imprenta, Ottaviano Petrucci, en 1501, publica "Harmonice Musices Odhecaton", con las primeras partituras impresas con tipos móviles, permitiendo la difusión masiva de la música escrita.

2.2.2 Cancioneros

En el siglo XIX, se transmitían los acordes folclóricos de forma oral, pero para el siglo XX, en la música popular (tango, rock, jazz) se empieza a usar la notación de acordes sobre la letra de la canción.

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

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 próximas líneas transcurrirán sobre aplicaciones web que son usadas actualmente por músicos, pero además de señalar solamente sus aportes, se hará 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 músico está tocando su instrumento, ajustado perfecto con el metrónomo, cuando llega al último acorde de la página (o de la pantalla) y lo obliga a soltar su instrumento para manipular el cancionero, perdiendo así el ritmo.

Las páginas suelen en su mayoría estar diagramadas en la pantalla pensado en maximizar el espacio para publicidad y no en la usabilidad del músico, por lo que ofrecen poca personalización en la vista. Esto hace que aunque muchas ofrezcan instrucciones sobre cómo se realizan los acordes en el instrumento, o algunos implementen un rudimentario autoscroll, sea poco visible para el músico.
De las que tienen contenido argentino, las principales son lacuerda.net, cifraclub y acordesweb.
Y claro, muestra solo los acordes. Algunas páginas de acordes, sin embargo, son páginas distintas!

2.3.2 Reproductores online de video y acordes

Estas aplicaciones web ofrecen videos con acordes sincronizados, como Chordify y Ultimate Guitar, ajustan la reproducción 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 llegó, hace rato...

Al momento de escribir esto, diciembre de 2025, los resultados del uso de la IA vienen avanzando enorme y exponencialmente.
Logran generar todo tipo de contenido, letras y partituras, audios y videos.
También permiten procesar audio para separar los instrumentos y transcribir partituras. Sobresale en ese sentido la aplicación Moises.ai y Spleeter.

2.5 Conclusiones

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 músicos.

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

El criterio de privilegiar el aspecto educativo y de usabilidad está ausente en los cancioneros online y en la mayoría 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 [2].

En la última vista, la vista de escenarios, los escribimos de igual modo que a las pruebas de aceptación en Reqnroll [3].

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.

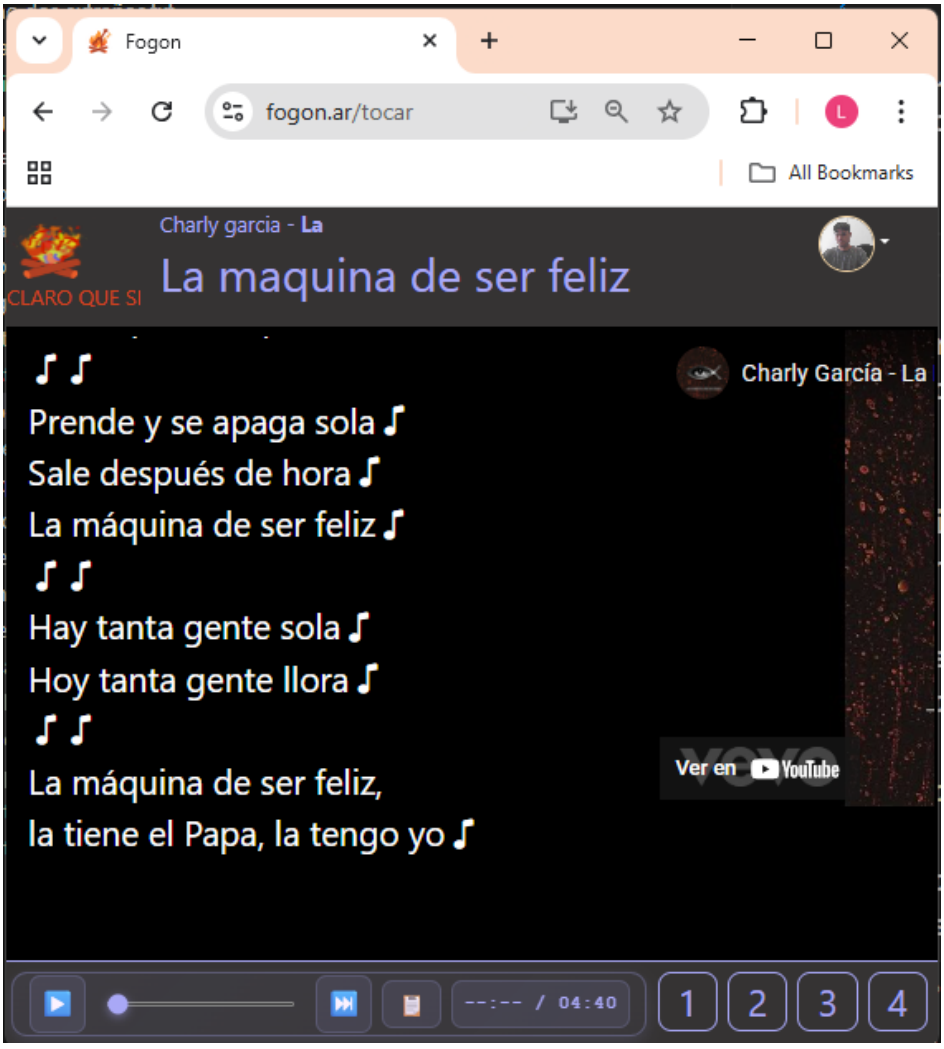

Figura 1: ”La maquina de ser feliz”, tocando con un video

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 página tocar muestra la vista según el músico:

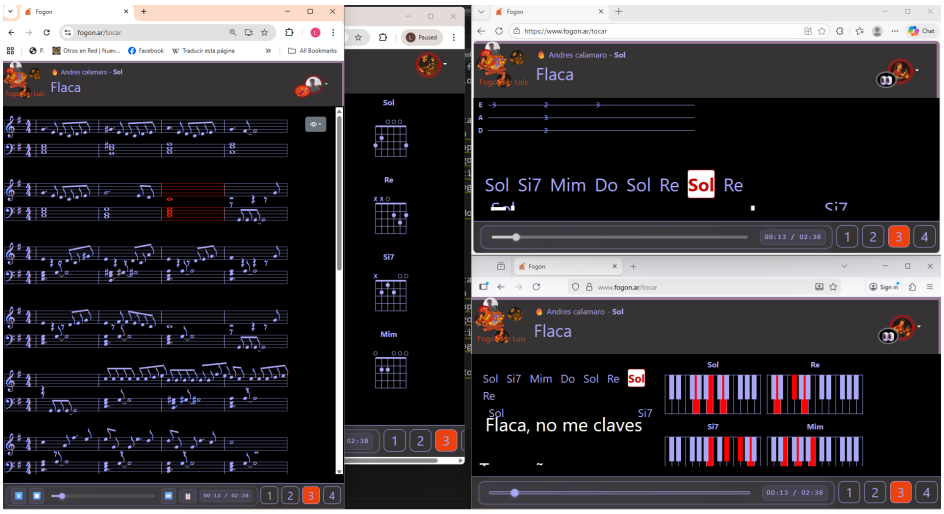

Figura 2: Página tocar con distintos instrumentos en varios exploradores

En la cabecera, muestra los datos de la canción y permite cambiar la vista y la configuración.

A lo largo de la pantalla, la vista particular del instrumento.

Debajo, el control de reproducción y el metrónomo.

icono-fogon Arriba a la izquierda, el icono del fogón muestra el estado general de la aplicación. Marca el pulso cuando están tocando y muestra los usuarios en la sesión.

Figura 3: Iconos para los estados: desconectado, conectado, logueado, o en sesión.

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

Vistas Para configurar su vista, cada músico accede desde el menú a “Ver”:

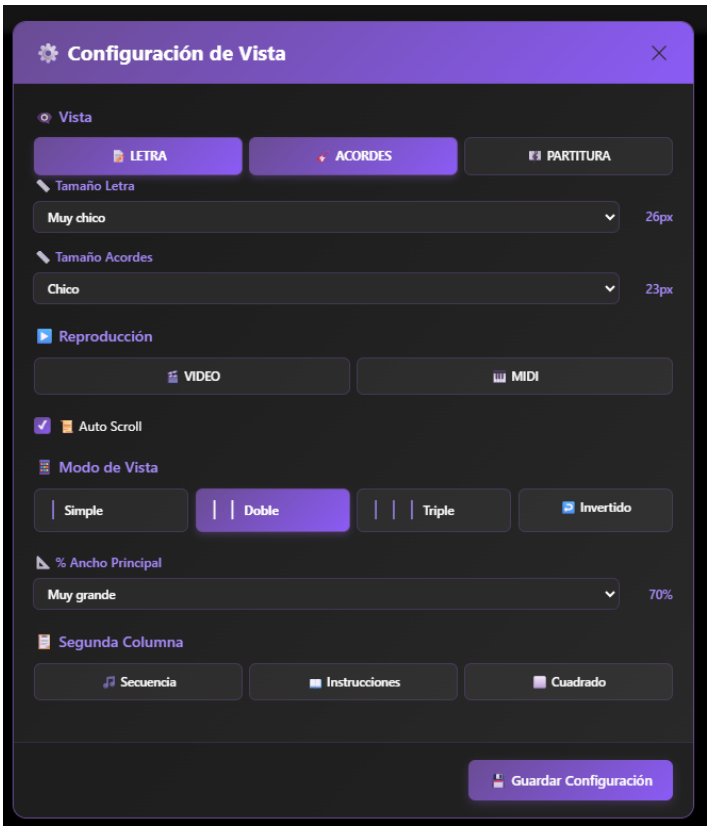

Figura 4: Configuración de la vista

Desde aquí puede elegir entre ver letras y/o acordes o la partitura. Además, puede ajustar el tamaño de la letra, los acordes y la partitura para ajustarla a cada dispositivo.

En la opción de reproducción, puede elegir acompañar la reproducción con un video o con un MIDI generado a través de las partituras.

En las últimas opciones, se ajusta la cantidad de columnas que se muestra en pantalla y si muestran instrucciones para el músico, 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 fogón, puede invitar a otros a unirse a la sesión colaborativa. Desde el menú, también, puede asignar distintos roles a cada usuario:

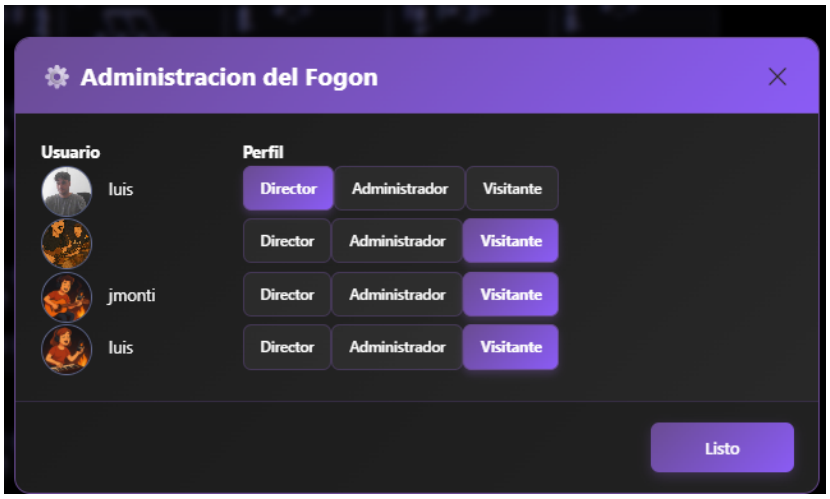

Figura 5: Configuración de roles en la sesión

Todos los usuarios ven la misma canción y el mismo compás, pero cada uno ve las instrucciones para su instrumento, en su dispositivo. Los administradores pueden cambiar la canción y controlar la reproducción. El director, es el único capaz de reproducir algún 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

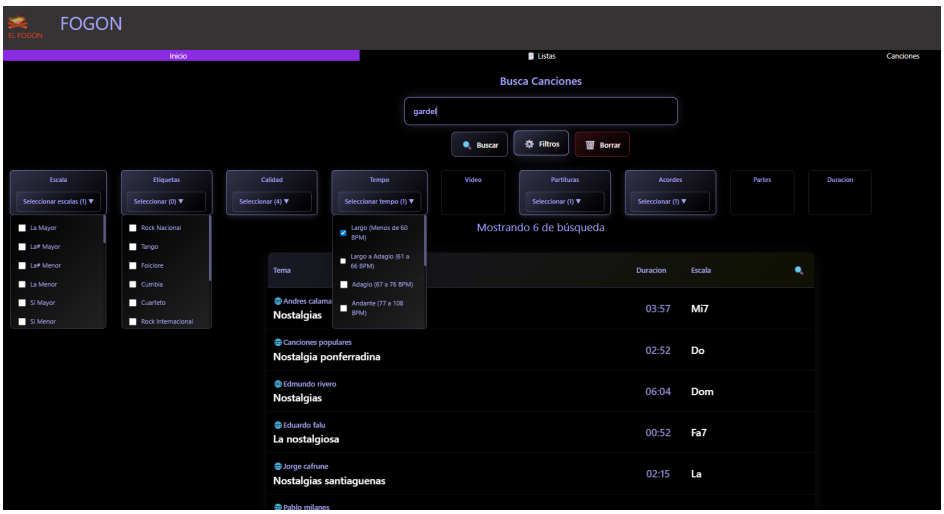

Figura 6: Busqueda de ”gardelçon los filtros desplegados de Escala, etiqueta y tempo

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

Se puede editar o una cancion, o crear una nueva desde cero, eligiendo sus principales caracteristicas como escala, secuencia armonica y tipo de letra.

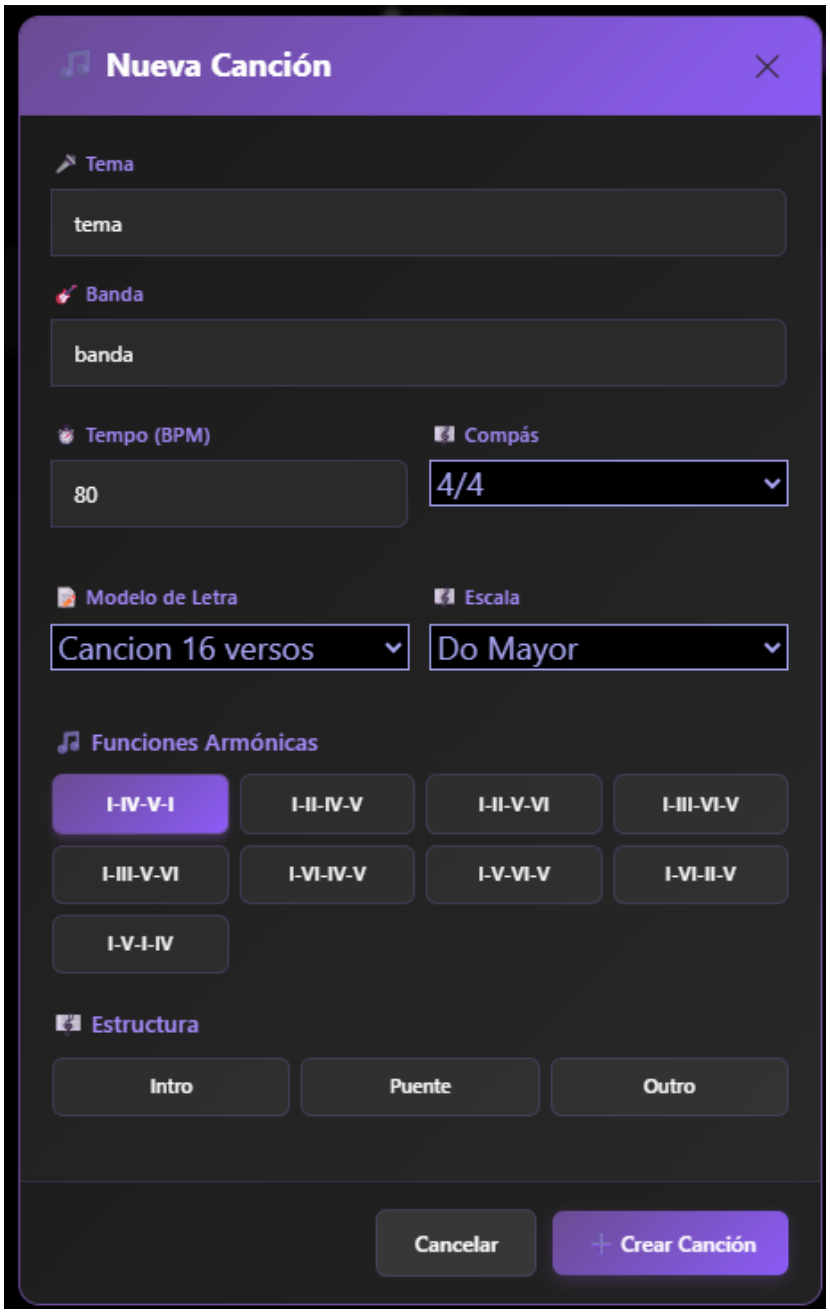

Figura 7: PopUp para crear una nueva cancion

En la pantalla de edicion, se puede modificar la letra, los acordes y las partes de la cancion.

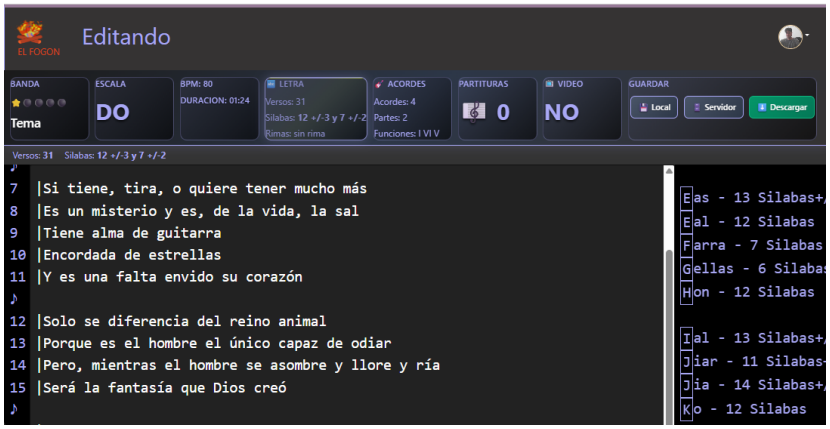

Figura 8: Ej de pantalla de edicion de letra

Editando la escala, es posible transponerla a cualquier otra tonalidad.

Tambien se puede agregar o editar pentagramas e importarlos desde archivos MusicXML.

3.1.5 Configurar

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

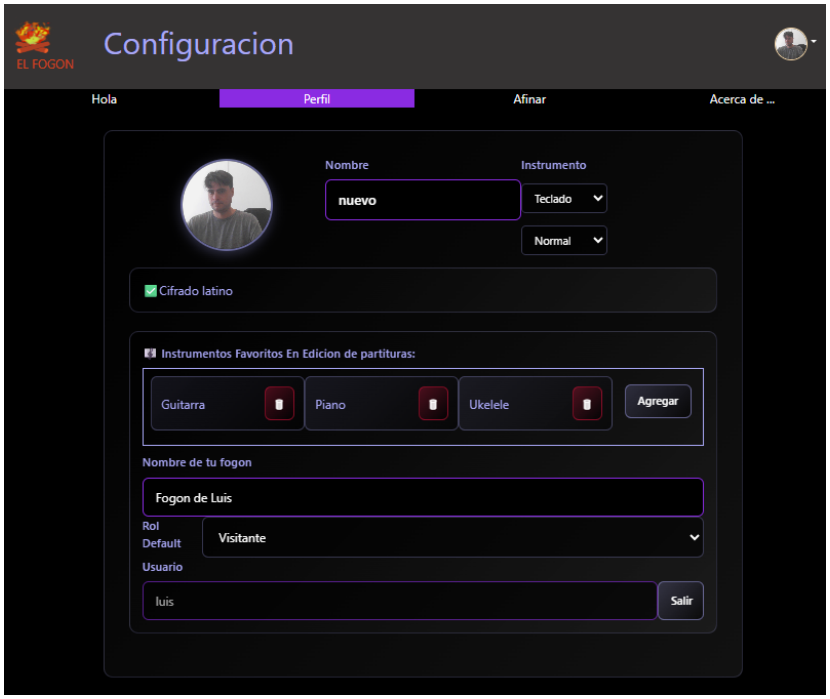

Figura 9: Pantalla de configuración del usuario

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...”

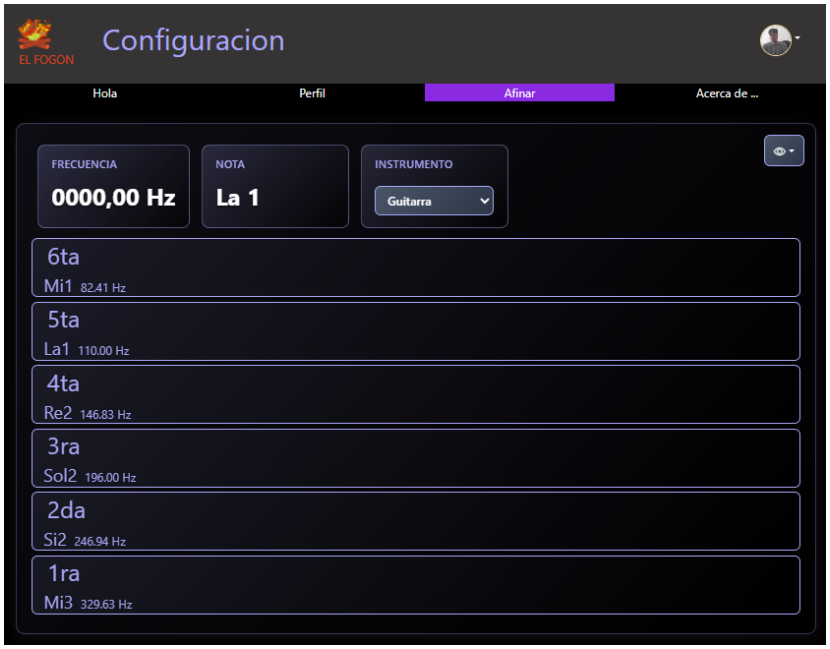

Figura 10: Pantalla de afinar instrumentos

3.2 Arquitectura del Sistema

Para describir la arquitectura del sistema, utilizamos el modelo de vistas 4+1 de Kruchten [2].

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].

3.2.1 Vista Lógica

Presentamos el modelo de datos, la clase principal es “Cancion”.

Luego la clase “Aplicacion”, que funciona como orquestador de los controles de la interfaz y la conexion con el backend, y por ultimo la clase “Reproductor”, que maneja la reproduccion de audio y la sincronizacion con los acordes y letras.

Canción Tiene dos propiedades de clase Letra y Acordes; además de propiedades título, artista, bpm, compás, etc.

La clase Acorde diseñada como una secuencia de partes, cada parte formada por una serie de acordes por compas. La propiedad ”secuencia” es la lista de partes como se tocan en la cancion.

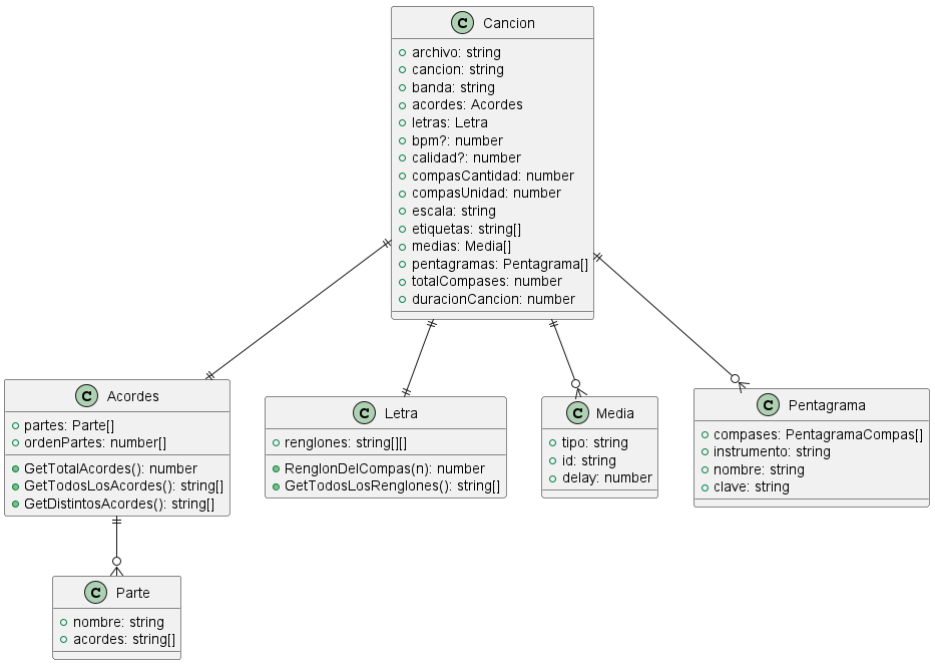

Figura 11: Diagrama de clases del modelo de Canción

Aplicación La clase Aplicacion orquesta entre el reproductor, la conexion y la interfaz de usuario. Maneja las clases ConexionManager para la comunicación con el servidor, Reproductor para la reproducción de la cancion, AutenticacionManager para el login/logout, y SesionManager para el manejo de sesiones colaborativas.

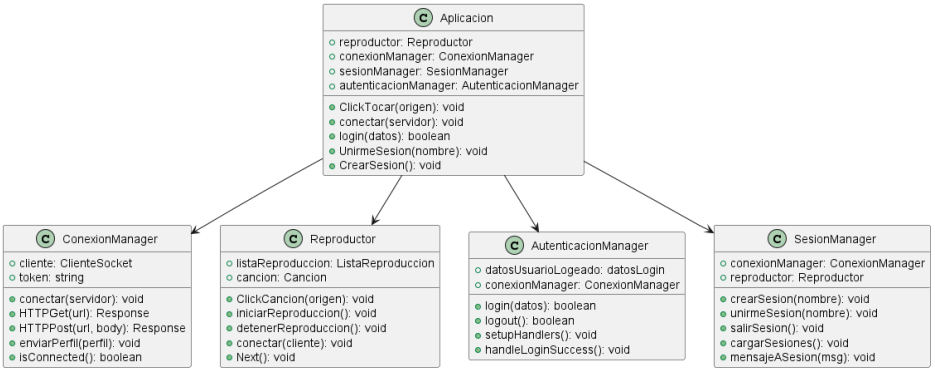

Figura 12: Diagrama de clases del sistema de Aplicación

Reproductor La clase reproductor administra la cancion y el momento actual. Implementa el patrón Strategy para reproducir en una sesion o en modo desconectado.

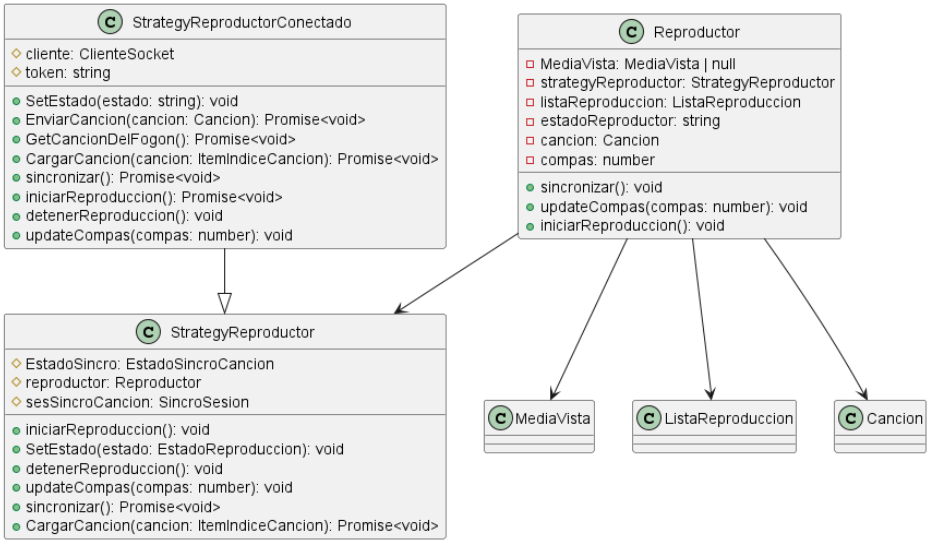

Figura 13: Diagrama de clases del Reproductor

En el siguiente capitulo esta explicado como interactuan estas clases.

3.2.2 Vista de procesos

Aqui repasaremos, con diagramas de secuencia, como la clase reproductor utiliza un Strategy para manejar la reproduccion en modo desconectado y en modo conectado.

Ejemplificamos con el proceso IniciarReproduccion, pero debe suponerse uno similar para DetenerReproduccion,CambiarCompas, CambiarLaListaDeCanciones, etc.

Luego, mostraremos la secuencia con la que logra sincronizar relojes con el servidor y entre clientes usando WebRTC.

IniciarReproduccion en modo local El reproductor delega en la estrategia ”StrategyReproductor”desconectada, que prepara la cancion en el cliente.

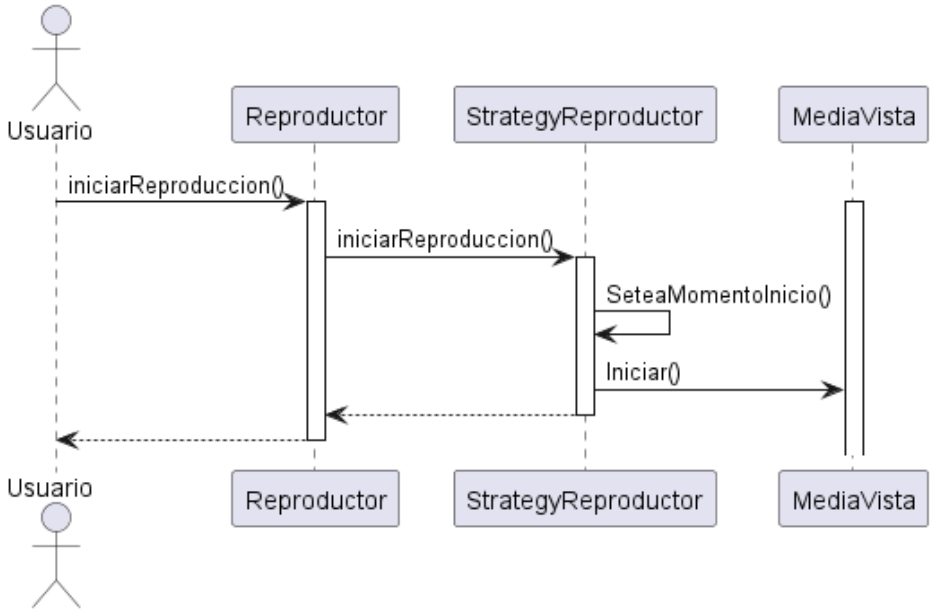

Figura 14: Secuencia de inicio de reproduccion en modo local

IniciarReproduccion en un Fogon Cuando se esta en una sesion, se manda al servidor la orden de iniciar reproduccion, y se espera la señal de inicio coordinado.

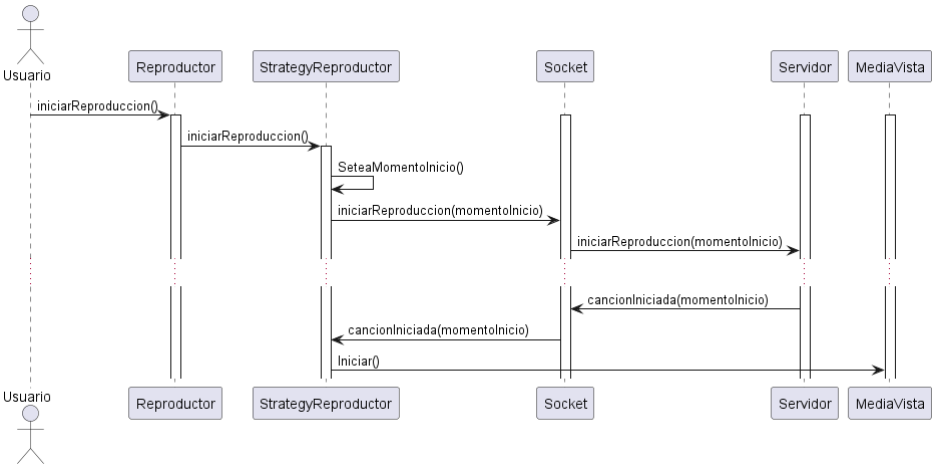

Figura 15: Secuencia de inicio de reproduccion distribuida: esta vez, la estrategia incluye al servidor y luego se inicia coordinadamente.

Sincronizar relojes Se muestran dos tipos de sincrozicanciones que se realizan en el sistema: por socket, cuando el usario se conecta al servidor, sincorniza su reloj con este. Cuando se conecta a un fogon, envia una oferta WebRTC para establecer un canal. La aplicacion puede sincronizarse con un usuario particular a pedido del servidor o del usuario: el primero porque los supone en la misma red WiFi, el usuario desde la configuracion. Vemos el intercambio de mensajes:

Figura 16: Secuencia de sincronizacion de reloj

3.2.3 Vista Física

En el diagrama de despliegue vemos que nuestra aplicacion cliente es Web y Progresiva, por lo que corre en el dispositivo del usuario, aunque busca en github pages el html, css y js necesarios; ademas de los indices.

El servidor en Go esta alojado en los servidores de Render y se conectan a una base MongoDB Atlas. Con algunos scripts, se actualizan canciones, por ejemplo para extraer la letra del video de YouTube.

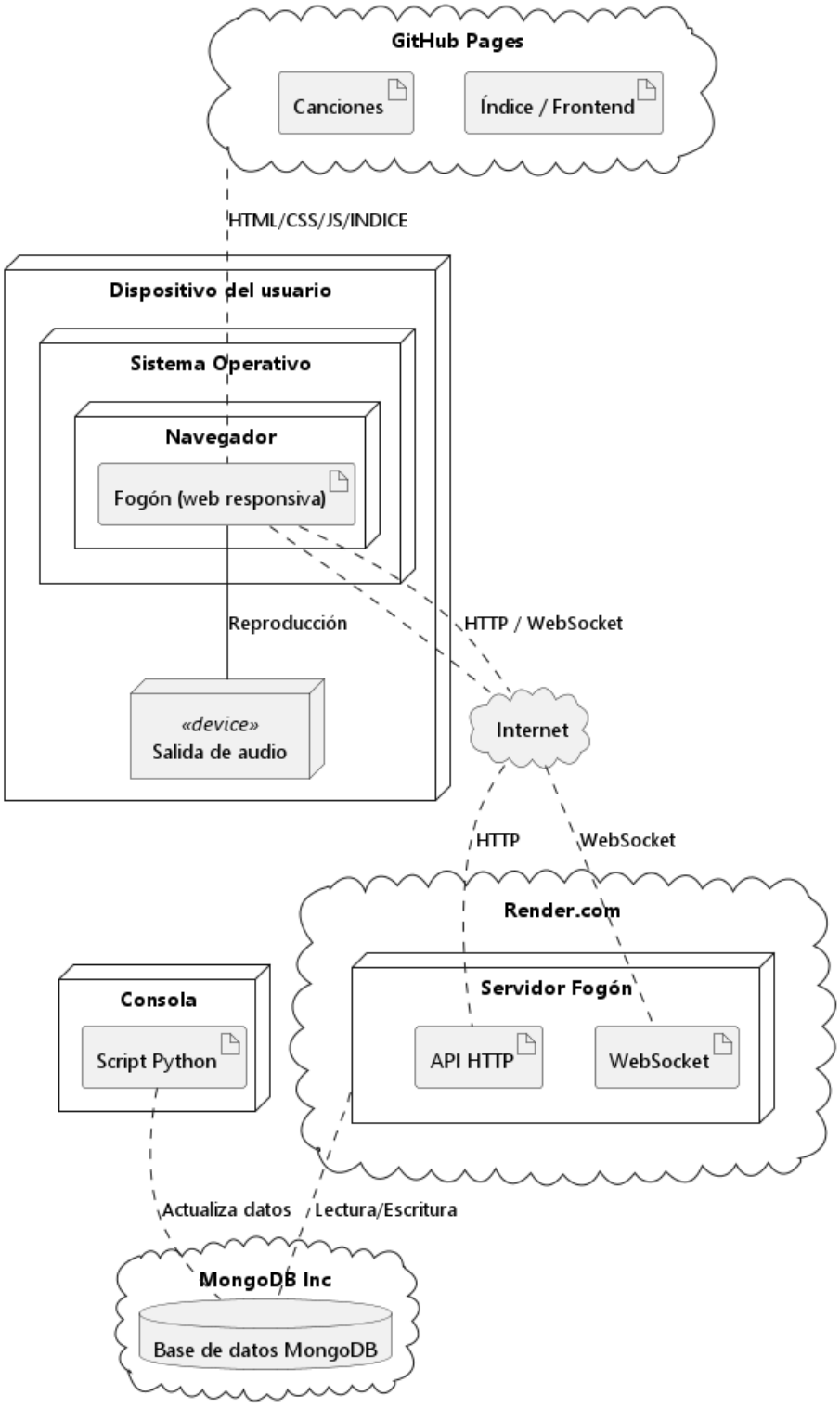

Figura 17: Diagrama de despliegue

3.2.4 Vista de Desarrollo

Estos son los principales paquetes del cliente, las vistas, los helpers y la aplicacion.

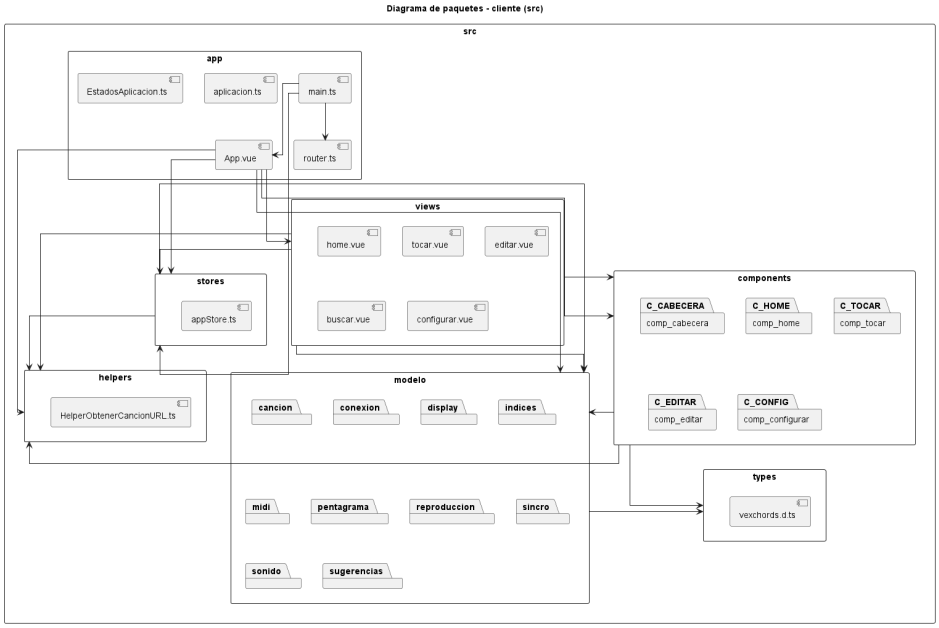

Figura 18: Diagrama de paquetes del cliente

En el servidor, los servicios acceden a los datos, a veces de modo directo o usando algunas reglas de negocios para listas o usuarios. Los controladores usan a los servicios para atender las peticiones y manipular los datos y la aplicacion, es creada desde el main.

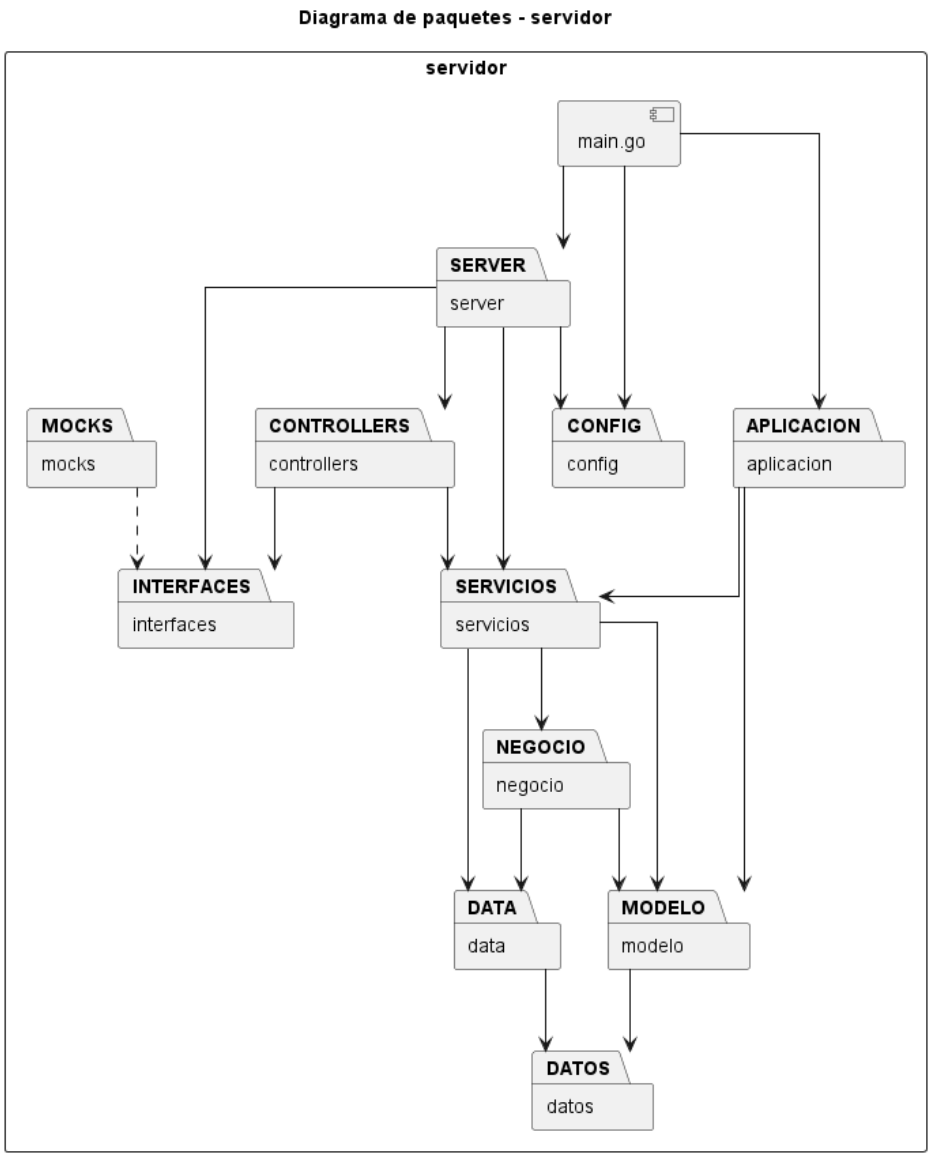

Figura 19: Diagrama de paquetes del servidor

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).

En este repositorio , estan desarrrolladas las pruebas automaticas con Playwright que validan estos escenarios: https://github.com/LuisWaldman/fogon-pruebas/

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: Buscar canciones
```

```
    El fogon permite buscar canciones por autor y por nombre
    pero tambien por caracteristicas tipicas como la escala, el tempo
       o la cantidad de acordes.

    Scenario: Busca paloma de calamaro
        Given Usuario va al fogon
        When busca "flaca calamaro"
        Then aparecen resultados relacionados con "flaca"
```

Contexto: Dos musicos quieren tocar una canción juntos desde sus dispositivos.

Listing 2: Escenario: Fogon

```
Feature: Sesiones colaborativas
    La principal particularidad del fogon es la de mantener sesiones
       compartidas
    El estado compartido es la cancion, la lista de canciones y los
       roles de cada integrante
    en la sesion (o fogon)

    Scenario: Dos usuarios se conectan
        Given "Usuario1" accede a la aplicacion
        And "Usuario2" accede a la aplicacion
        And "Usuario2" inicia un fogon
        And "Usuario2" carga la cancion "Homero"
            When "Usuario1" se une al fogon de "Usuario2"
        And "Usuario1" va a tocar
        Then "Usuario1" ve la cancion "Homero" en reproduccion
        And "Usuario2" y "Usuario1" reproducen la cancion de modo
           calibrado
```

3.3 Tecnologías Utilizadas

Para el desarrollo de .[El] Fogón"se seleccionó un conjunto de tecnologías modernas con el objetivo de construir una aplicación robusta, escalable y de alto rendimiento.

3.3.1 Frontend

- **Vue.js:** Se eligió como el framework principal para la interfaz de usuario por su curva de aprendizaje amigable, su reactividad y su ecosistema de componentes. Permitió crear vistas complejas e interactivas de manera eficiente.

- **TypeScript:** Se utilizó en todo el frontend para agregar tipado estático a JavaScript, lo que

mejoró la robustez del código, facilitó la detección temprana de errores y mejoró la experiencia de desarrollo.

- **Vite:** Como herramienta de construcción y servidor de desarrollo, Vite ofreció un arranque casi instantáneo y recarga en caliente (Hot Module Replacement) extremadamente rápida, agilizando significativamente el ciclo de desarrollo.

- **VexFlow:** Es la librería clave para el renderizado de partituras. Aunque su integración fue compleja, era indispensable para cumplir con los requisitos de visualización de notación musical estándar.

3.3.2 Backend

- **Golang:** Se seleccionó para el desarrollo del servidor por su excelente rendimiento, su concurrencia nativa (ideal para manejar múltiples conexiones WebSocket y WebRTC) y su simplicidad para crear APIs RESTful y servicios multihilo.

3.3.3 Comunicación y Sincronización

- **WebSocket:** Es la base para la comunicación en tiempo real entre el cliente y el servidor, permitiendo la gestión de sesiones y el envío de comandos de control.

- **WebRTC:** Se empleó para la sincronización de alta precisión de los relojes entre clientes, un requisito crucial para la reproducción musical distribuida y la compensación de jitter.

3.3.4 Base de Datos

- **MongoDB:** Se utilizó como base de datos NoSQL para almacenar la información de usuarios y canciones, gracias a su flexibilidad de esquema que se adapta bien a la estructura de los datos de la aplicación.

3.3.5 Pruebas y Calidad de Código

- **NUnit y Playwright:** Se combinaron para realizar pruebas de aceptación de punta a punta, automatizando la interacción con la interfaz de usuario en un navegador real para validar los flujos de la aplicación de manera integral.

- **Reqnroll:** Se adoptó este framework de BDD para escribir los escenarios de prueba en un formato legible (Gherkin), facilitando la colaboración y asegurando que el software se comporte como se espera desde la perspectiva del usuario.

3.3.6 Gestión y Despliegue (DevOps)

- **Git y GitHub:** Se utilizaron para el control de versiones del código y la gestión del proyecto, incluyendo el seguimiento de tareas mediante GitHub Projects.

- **Render.com y Fly.io:** Estas plataformas de Platform as a Service (PaaS) se usaron para el despliegue continuo y el alojamiento tanto del frontend como del backend, facilitando la publicación de nuevas versiones de forma automática.

3.3.7 Otras Herramientas

- **Modelo Claude:** Modelo de IA

4 Metodología Aplicada

Describiremos la metodología aplicada y la planificacion inicial, despues comentaremos los cambios al plan inicial que se acordaron durante el desarrollo del proyecto.

4.1 Metodología del trabajo

El trabajo tendrá 6 hitos en los que se publicará la aplicación. Para alcanzar cada uno habrá entregas incrementales periódicas, la mayoría de una sola etapa y algunas de 2. Gestionaremos las tareas utilizando el siguiente proyecto de GitHub: `https://github.com/users/LuisWaldman/projects/4`. Detallamos a continuación las tareas que integran a cada hito.

4.2 Plan

4.2.1 Tocar

Finaliza con el sitio desplegado en `http://www.fogon.ar/` con integración continua. Permite tocar algunas canciones.

Tareas	**Duración (horas)**
SetUp inicial: armar el repositorio, crear la aplicación Vue.js + TypeScript, los módulos de prueba y el despliegue automático	12
Definir JSON para canciones y armar ejemplos	4
Armar estructura de la aplicación	4
Armar menú	2
Controlador de tiempo	8
Armar pantalla tocar acordes	4
Armar pantalla tocar letra	4
Armar pantalla tocar letra y acordes	12
Total	**50**

4.2.2 Sincronizar

Varios usuarios pueden unirse a una sesión y reproducir juntos una lista de canciones.

Tareas	Duración (horas)
Definir protocolo	8
Login de usuarios con Google (extendible)	8
SetUp inicial del servidor	10
Administración de conexiones y estructura	30
Permitir crear una sesión desde la aplicación web	6
Permitir unirse a una sesión desde la aplicación web	6
Permite ver el estado de las sesiones desde la configuración de la aplicación web	12
La canción se reproduce con el ritmo de la sesión cuando el usuario está conectado	12
Los distintos usuarios pueden mandar comandos al servidor que se reflejan en todos los conectados	12
Permite que la sesión siga viva cuando se desconecta el Admin, asignando el rol	4
Total	**108**

4.2.3 Editar

Agrega la administración de usuarios, que pueden editar y compartir las canciones.

Tareas	Duración (horas)
Pantalla de configuración	6
Administración de usuarios	6
Configuración y despliegue de una base de datos MongoDB	6
Permitir obtener la canción de distintos repositorios: solicitud HTTP de canción pública en el sitio, IndexDB o en el servidor, canciones propias o de otros usuarios	4
Permite cambiar datos básicos de la canción	8
Permitir modificar la secuencia de acordes (el orden en que se reproducen las partes)	4
Permitir modificar los acordes de las partes: unir, cambiar, etc.	10
Permite modificar las partes: agregar, eliminar, combinar o dividir	10
Permite modificar la letra en la vista de letras y acordes	20
Permite guardar la canción en el servidor, el IndexDB o el disco duro	10
Permite cambiar la escala (cambia todos los acordes de la canción)	8
Total	**92**

4.2.4 Buscar y listar

El usuario puede buscar canciones en distintos repositorios y armar y compartir listas de reproducción.

Tareas	Duración (horas)
Armar índices	6
Armar filtros	6
Obtener canciones mediante web scraping de internet	12
Mostrar búsqueda en un gran repositorio	4
Mostrar listas	8
Permitir guardar/editar listas	8
Definir una lista como lista de reproducción actual	2
Desde “Tocar” reproducir la lista de canciones actual	6
Total	**52**

4.2.5 Tocar y editar partituras

Muestra las partituras con VexFlow, permite editarlas y cargarlas desde archivos XMLMusic.

Tareas	Duración (horas)
Agregar a JSON de la canción la posibilidad de guardar partituras para distintos instrumentos	6
Permitir desde la aplicación web definir un instrumento default para que te muestre la partitura	3
Mostrar partituras con VexFlow	25
Permitir editar partituras con VexFlow	50
Permitir agregar partituras desde archivos en el formato XMLMusic	16
Generar ejemplos de partituras	2
Total	**102**

4.2.6 Más instrumentos y detalles

Tareas	Duración (horas)
Reparar bugs de versiones anteriores	10
Vista de acordes con dibujos de cómo sería en la guitarra	15
Vista de tablaturas de riffs en guitarra	15
Vista para armónica	15
Reproducir las partituras en formato MIDI	20
Total	**75**

4.3 Herramientas Utilizadas

- El Visual Code como entorno de desarrollo integrado (IDE) principal para escribir y depurar el código.
- Vite como herramienta de construcción y empaquetado del proyecto.
- Git y GitHub para el control de versiones y la colaboración en el desarrollo del proyecto.
- El modelo Claude Sonnet 4.5 para desarrollo de pruebas, de código y de vistas.
- Jira para la gestión de proyectos y seguimiento de tareas.

4.4 Desarrollo de la aplicación

El desarrollo comenzó el 11 de Julio de 2025 con la aprobacion de la propuesta de trabajo.

En la primer entrega, se contruyó el repositorio con integracion continua, de modo que para cada commit se ejecutaban pruebas y verificacion de estilo de codigo. Cada branch al main, genera ademas el despliegue de la aplicación.

En las siguientes entregas, la de la sesion sincronizada y la de la edicion, vimos necesario adelantar las tareas relativas a pentagramas.

La integracion con VexFlow efectivamente fue compleja y adelantar su desarrollo termino siendo una buena decicion.

En la ultima entrega planificada, agregamos listas y busquedas tanto en el servidor como en la IndexedDB del navegador.

Durante todo el desarrollo, sentimos la necesidad de agregar nuevos features, que desarrollamos como una nueva entrega:

Tareas	**Duración (horas)**
Sincronizar relojes por WebRTC	20
Mostrar Acordes de notas en cifrado español	10
Afinador	10
Poder sincronizar la letra de las canciones con videos de YouTube	25
Vista de acordes en teclado	8
Actualización automática de la aplicación	2
Total	**75**

El 4 de Diciembre de 2025 se realizó la entrega final, con el sitio desplegado en `http://www.fogon.ar/` y con todas las funcionalidades mencionadas en el plan inicial y las adicionales.

5 Experimentación y Validación

Para la validacion y el debug de la aplicación, se crearon distintas herrmientas; la mayoria de ellas se acceden desde el perfil, accediendo al ”modo desarrollador”. Desde ahi se puede modificar el servidor al que se conecta y ver los detalles de la sinconización y las sesiones.

Veremos en el proximo capítulo como los utilizamos para validar la sincronizacion y tomar algunas deciciones de diseño.

Luego, repasaremos las pruebas unitarias y de aceptación que se desarrolloron durante la aplicación y despues de la aplicación.

5.1 Calibración

Vemos los datos de calibración del reloj en desarrollo:

Figura 20: Calibración del reloj en desarrollo

Muestra la hora local y la del servidor. En el segundo cuadro la diferencia entre ambos y el delta de error. Presionando en la lupa, muestra el detalle de la medicion. En este caso, 10 pedidos con su round trip time y la diferencia entre el reloj local y el del servidor. De ellos, obtiene los 6 mejores y calcula el promedio. En mi maquina, claro, anda perfecto. Menos de 4 milisegundos de margen de error, en

cambio, estos son los datos con la aplicacion desplegada en github y onrender.com:

Figura 21: Calibración del reloj con el servidor en render.com

114.14s de margen de error es mucho mas que lo esperable, teniendo en cuenta que el humano empieza a notar el delay a partir de los 20ms. Buscamos, entonces, un servidor especializado en la velocidad: fly.io:

Figura 22: Calibración del reloj con el servidor en fly.io

Vemos que con este servidor alcanzamos un tolerable margen de 22 milisegundos. Hubo que desarrollar una herramienta para calibrar dispositivos en una misma red local, una vez que inician un fogon. Cuando esto ocurre, se conectan con WebRTC, alcanzando estos resultados:

Figura 23: Calibración del reloj RTC

Como vemos, conectados por WebRTC, volvemos a calibrarnos con otro dispositivo con margenes menores al milisegundo.

De este modo, logramos evitar la dependencia con un proovedor de hosting por su latencia.

5.2 Pruebas Unitarias

Las clases del negocio fueron testeadas con vitest, muchas reglas y algoritmos propios de la musica y de la poesia, fueron resueltos guiando el desarrrollo con pruebas unitarias.

Lo usamos para calcular los acordes de una escala:

Listing 3: Prueba unitaria de la clase MusicaHelper

```
import { MusicaHelper } from './MusicaHelper'
import { describe, it, expect } from 'vitest'

describe('MusicaHelper', () => {
  it('Notas de la escala', () => {
    const helper = new MusicaHelper()

    [...]


    expect(helper.GetAcordesdeescala('Am')).toEqual([
      'Am',
      'Bdim',
      'C',
      'Dm',
      'Em',
      'F',
      'G',
    ])
    [...]
  })
})
```

Para contar las silabas de un verso:

Listing 4: Prueba unitaria de la clase SeparadorSilabasRima

```
  it('Pruebo contar silabas de versos famosos', () => {
    expect(
      sepSilabas.GetNroSilabasVerso(
        sepSilabas.getSilabasPalabra('Sonye que el rrio me hablaba'),
      ),
    ).toEqual(7)
  })
```

Para calcular los acordes del ukelele:

Listing 5: Prueba unitaria de la clase SeparadorSilabasRima

```
1
2     it('debe devolver el acorde E correctamente', () => {
3       const acorde = AcordesUkeleleHelper.getAcorde('E')
4       expect(acorde.acorde).toBe('E')
5       expect(acorde.cejilla).toBe(0)
6       expect(acorde.cuerda[0]).toBe('2')
7       expect(acorde.cuerda[1]).toBe('4')
8       expect(acorde.cuerda[2]).toBe('4')
9       expect(acorde.cuerda[3]).toBe('4')
10    })
```

5.3 Pruebas Aceptación

Las pruebas de aceptación se desarrollaron con NUnit y Playwright. Como mencionamos en las conclusiones, estas pruebas no fueron parte del proyecto. Elaboré algunas para esta presentación, y debería hacer mas segun el proyecto crezca.

Quedán en el repositorio https://github.com/LuisWaldman/fogon-pruebas/

6 Conclusiones

Resumen final del trabajo, destacando los objetivos alcanzados y el valor del sistema desarrollado.

6.1 Desarrollos Futuros

El sistema está pensado para la escalabilidad, en modo "desarrollador", en las opciones de configuración se puede definir el servidor al que se conecta. Si el fogón tiene éxito mundial, nos bastaría con agregar más servidores distribuidos por región e implementar un paso previo en el que el cliente se conecta con un balanceador de carga que lo redirige al servidor más cercano.

El diseño de la aplicación, y principalmente Vue.js, permite agregar nuevas vistas sincronizadas. Podríamos agregar alguna que muestre el teclado con las notas bajando y alguna vista para bateristas.

Otra funcionalidad que quedó pendiente, es la de que la aplicación no avance el tiempo según transcurre, sino según escucha lo que escucha que el músico está tocando. Así, podría ayudar "esperando" a un aprendiz.

Nos queda agregar algún modo de editar ritmo como acompañamiento, que sumado a los pentagramas de VexFlow, permitiría editar canciones completas.

Aún hace falta sumar más repertorio. Durante el desarrollo se crearon varias canciones y algunos algoritmos para obtener los acordes desde un cancionero, y las letras de las canciones desde Youtube. Igual, hace falta crear más canciones con letra + acordes + partitura + video que permitan el uso completo de la aplicación.

6.2 Lecciones aprendidas

Durante el desarrollo del proyecto, el año 2025, el uso de la IA en programación pasó de ser recomendada a inevitable. A fines de septiembre, cuando empezaba a programar una función, ya aparecía sugerida, a veces mejor de lo que pensé, a veces antes de que lo piense.

Como todos, caí en la tentación de hacer "no-code" en algunas funcionalidades: traspasar tonalidad de canciones, contar versos en poemas, pasar el audio a notas, etc. Pero cuando algún problema empezaba a ser complejo, una acción de la IA deshacía lo que había hecho anteriormente. Lo mismo pasaba con los humanos y ya teníamos una solución: guiar el desarrollo con pruebas.

Durante la construcción del proyecto el modelo claude sonet 4.5, guiado por pruebas (que hasta a veces escribía con el mismo), intervino en el desarrollo de la mayoría de las funcionalidades y en la corrección de bugs.

Sobre el final del proyecto, cuando este tenía distintas vistas y estados, se hizo necesario un modo de probar la aplicación en su conjunto. Sobre el final del desarrollo, armé test punta a punta con Playwright; pero si tuviera que rehacer el proyecto en 2026, lo haría desde el principio con pruebas de integración, o sea: desarrollo guiado por comportamiento.

La IA va a seguir tomando lugar: en este momento que escribo, si dejo de tipear un ratito, una IA me sugiere cómo terminar la frase. El profesor que lea algún trabajo profesional seguro lo va a pasar por un filtro con IA y el alumno que lo presenta también. Esto nos deja a todos un elevado nuevo piso y nos posibilita alturas mayores.

Aves de mal agüero anuncian un cambio en el estado del arte cada 1 minuto, porque asumen al humano totalmente ausente de las decisiones tecnológicas. Pero si el humano quiere seguir guiando el desarrollo, lo va a tener que hacer a través de una arquitectura clara y de pruebas que validen el comportamiento.

6.3 Conclusiones

El proyecto fue ambicioso, no solo quería que el proyecto sea novedoso, sino que el conjunto de tecnologías utilizadas sea el ideal y que reflejara lo que aprendí en la facultad.

Con respecto a las herramientas, el framework vitejs, con el código regulado por “Lint” y pruebas unitarias en una integración continua que termina desplegándose en el fogon.ar; terminó resultando ser una excelente elección. Sobre el servidor, Golang hizo fácil crear APIs y procesos multihilos para atender a los clientes; y tanto en Render.com como Fly.io, pudimos hacer despliegues automáticos.

Sobre lo aprendido en la facultad, En Técnicas de Diseño, dejamos de “solo programar” para empezar a pensar los sistemas por su arquitectura y a ordenar los problemas por los patrones que lo resuelven. Luego, en Introducción a Sistemas Distribuidos y en Sistemas Distribuidos entendimos qué es un protocolo, cómo calibrar relojes, qué es y cómo se elige un proceso líder.

Esto lo utilicé para pensar los desafíos principales en la aplicación. Para el pentagrama en VexFlow, por ejemplo, decidimos usar el patrón Facade en vez de Proxy, y al revés para administrar el video; las múltiples vistas son reutilizables y extensibles porque están separadas del modelo y del controlador; El estado compartido del Fogón, es pensado como “estado compartido” porque toda la aplicación está pensada en los términos aprendidos en Sistemas Distribuidos.

Además, desarrollé otras herramientas para medir y validar la exactitud de la calibración, lo que muestra un rigor científico característico de la academia.

Por todo esto creo que el fogon.ar suma a los conocimientos adquiridos en FIUBA una mirada original, llevada a cabo con herramientas del estado del arte actual, con rigor académico y científico, y con un resultado que sobrepasa los objetivos.

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].