Universidad Zaragoza

1542

Trabajo Fin de Máster

Desarrollo de una red social y herramientas para cantantes mediante una aplicación Android

Creation of a social network and tools for singers through an Android application

Rouffineau Frédéric

Directores

José Ramón Beltrán Blázquez
Víctor Viñals Yúfera

Escuela de Ingeniería y Arquitectura

Mayo 2016 - Febrero 2017

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Frédéric ROUFFINEAU,

con nº de DNI Y4399418S (NIE) en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Máster, (Título del Trabajo) Desarrollo de una red social y herramientas para cantantes mediante una aplicación Android (Creation of a social network and tools for singers through an Android Application),

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 1 de febrero de 2017

Fdo: Frédéric

Agradecimientos

Un sincero agradecimiento para mi director de trabajo J.R Beltrán que mostró un auténtico interés por el trabajo y con quien he podido compartir varias cosas acerca de la música. No se puede olvidar a mi ponente V.Y. Viñals cuyo soporte ha sido constante a lo largo del máster. Por último, este trabajo está dedicado a todos mis amigos músicos y no músicos, y a mis colegas de trabajo en Hiberus del departamento Magento, que facilitaron mucho la realización de este proyecto y de esta memoria, por su presencia, comprensión, flexibilidad, y apoyo.

Resumen

Universidad de Zaragoza- Escuela de Ingeniería y Arquitectura
Máster en Ingeniería Informática

Desarrollo de una red social y de herramientas para cantantes mediante una aplicacion Android

por Frédéric ROUFFINEAU

Hoy en día, empezar el canto no es cosa fácil sin formar parte de una familia de músicos, la falta de confianza y de un seguimiento apropiado puede impedir el disfrute de este sencillo placer. Este proyecto, es una aplicación para Android llamada Singvibes. Tiene como próposito ofrecer herramientas que ayuden a cantantes o personas que quieran empezar a cantar. Singvibes combina una red social con herramientas de análisis de sonido y de voz usando el algoritmo Fast-Lifting-Wavelet-Transform basado en las wavelets de Haar, que proporciona la información en tiempo real de la afinación del canto. La red social tiene como propósito poder compartir las grabaciones y encontrar a otros cantantes que pueden compartir su opinión, aportar una ayuda y un oído humano imprescindible para mejorar en canto, creando un espacio positivo.

Para realizar este doble proyecto (aplicación de análisis vocal cliente Android y red social basada sobre un servidor desarrollado integralmente personalmente) fue necesario el uso de una gran parte de los conocimientos enseñados en Ingeniería Informática. Eso incluye los campos siguiente:

- Adquisición y procesamiento de una señal digital en tiempo real
- Bases de datos (relacionales): diseño, esquema en estrella, data warehouse, flujos de datos, MySQL y SQLite
- Administración de servidor y Cloud, estando el servidor desplegado en el cloud de Amazon.
- Arquitectura distribuida: fueron necesarios varios patrones de diseño para implementar correctamente el paradigma cliente->servidor y gestionar la concurrencia entre usuarios.
- Redes y sistemas: tomar medidas para asegurar disponibilidad, seguridad, fiabilidad, coherencia y consistencia de los datos. Limitar tanto como sea posible el tráfico mediante un cacheado de la base de datos en los dispositivos clientes.
- Conocimientos en diseño y desarrollo de aplicaciones Web: webservicios, APIs, contenido estático y dínamico y paradigma Modelo-Vista-Controlador
- Programación orientada a objetos: lado cliente y lado servidor
- Calidad en el desarrollo: prestando mucha atención a la modularidad, extensibilidad, con uso de sistema de versioning (git)
- Computación de altas prestaciones: paralelización, multi-threading
- Sístemas empotrados: uso del framework android y uso de sensores (micrófono) para procesar y comunicar datos

Este proyecto está ahora en estado Beta, con las funcionas centrales implementadas, pero con varias funcionalidades adicionales que se deben implementar antes de la publicación en el Google Play (plataforma de aplicaciones para Android). Se introducirá también un plan de explotaciones y una presentación de las áreas de mejoras.

Abstract

Universidad de Zaragoza- Escuela de Ingeniería y Arquitectura
Máster en Ingeniería Informática

Creation of a social network and tools for singers through an Android application

by Frédéric ROUFFINEAU

Singing is not an easy thing to master for someone wihout a musical background. The lack of trust and relevant mentoring can quickly put off people from enjoying this simple pleasure. This project, an application for Android called Singvibes, hopes to provide a set of tools for beginners or advanced singers, accompanying their training and opening them new prospects by obtaining some advice from other people just like them through a collaborative-sharing system. Indeed, Singvibes aims to combine a social network with a training companion doing some real time voice analysis using the embedded phone microphone sending the signal to a Fast Lifting Wavelet Transform algorithm (with the Haar wavelet) in order to provide useful information to the singer.

A wide range of Computing Engineering knowledge has been required to develop this double project:

- Digital signal real-time processing and reconstruction
- Relational databases: design, star-shaped schemas, data warehousing, data stream processing, MySQL and SQLite systems
- Server and Cloud management
- Distributed Architecture: client-server paradigm and underlying notions related to it: concurrent connections, session management, work load
- Network and distributed systems: take measures to ensure a decent enough level of availability, safety, security, reliability, consistency and coherency of the data while exchanging various types of data, and limiting the exchanged volume as much as possible
- Knowledge about the internet and web applications: webservices, APIs, dynamic and static content management, and structured code production - front end controller pattern and Model-Viwe-Controller
- OOP (Object-Oriented-Programming) - server and client side using various abstractions and packages to make some reusable and modular code
- Quality software development: best practices through an extensive documentation of what it is currently done by professionals all across the world
- High-demand computation: parallel execution, multithreading
- Embedded systems: Android framework for mobiles, allowing the use of sensors, real-time processing, display and exchange of information with a remote system

This projectly currently lies in its beta version. The core funcionalities are working, but before pushing up to the Google Play platform, it needs some more maturing and improvements which will be described at the end of this report. A long-term business plan including advertisment, sponsoring and a ÿou get what you paid for¨policy have been thought and will be executed as soon as possible.

Índice general

1 Introduction

1.1. Objetivos

1.1.1. Singvibes: un 'oído virtual' para cantantes

Hoy en día, se puede oír con bastante frecuencia afirmaciones del tipo siguiente: 'el canto, es innato'. O 'tienes la voz o no la tienes', o 'sé que canto muy mal', o 'no tengo oído'. Pero hay muchas personas a quién les gusta cantar, aunque sea algunos minutos para animar el día o la ejecución de tareas rutinarias. Hay un conjunto de creencias que hace que dentro de todas las personas interesadas por el canto, pocas lo intentan de verdad pensando que les falta oído o talento para hacerlo. Sin embargo, cualquier cantante profesional sabe la cantidad de horas que hay que practicar para cantar afinado y con un timbre de voz a veces muy lejos de la voz 'natural' que cada uno tiene.

De eso surgió la idea de crear una aplicación que incite a la gente a probar el canto, y ver que también pueden emitir notas, más o menos afinadas, igual que los demás, proveyendo un feedback en tiempo real objetivo (la comparación de la nota cantada con la nota más cerca de un piano por ejemplo).

La aplicación que se va a presentar, Singvibes, está pensada como un oído artificial capaz de compensar y entrenar al usuario para que cante mejor, y más afinado, reforzando su nivel de conciencia mediante la grabación por el micrófono del teléfono usando simultáneamente técnicas de análisis de sonido en tiempo real.

1.1.2. Una oportunidad de encontrar a otros músicos

Para cantar bien, no basta solo con cantar las notas afinadas y en ritmo. Hay que tener una cierta comprensión de la música, una cierta práctica y conocimiento de su propio instrumento (la voz), cuya complejidad viene dada, en gran medida, por su carácter individual. Entonces, el análisis del sonido, por muy potente que sea, no será suficiente para mejorar el canto de alguien. Hay también que enfrentarse al público y a sus opiniones, y eso puede generar ansiedad y miedo.

Así que, una manera concreta de mejorar es escucharse y obtener también la opinión de los demás, a todos los niveles. Los principiantes tienen la ventaja de compartir la ansiedad y la decepción que uno puede experimentar a la hora de empezar a cantar y darse cuenta de que la grabación no es buena. Y los más experimentados pueden compartir su experiencia y obtener satisfacción de ello. Aquí está el segundo objetivo de Singvibes: crear un espacio positivo de encuentro entre gente que comparte la misma afición, donde se puede obtener una opinión cuando es necesaria, que añade al oído virtual objetivo, un oído humano subjetivo fundamental para ganar soltura en canto.

1.2. Estado del arte

1.2.1. Bases teóricas y científicas

Cumplir los objetivos mencionados antes no se puede hacer sin el trabajo enorme que hicieron muchos grandes matemáticos y físicos sobre el sonido y la música, como Helmholtz y sus resonadores o Fourier y sus series.

En efecto, la física ondulatoria está bastante desarrollada y hay muchos modelos y ecuaciones sobre las ondas y, por tanto, el sonido que son entendidas, establecidas y consideradas como buenas aproximaciones de la realidad (la definición de un modelo) desde hace mucho tiempo.

Las nociones usadas para construir el detector de tono serán detalladas más adelante.

1.2.2. Ejemplos de aplicaciones ya existentes

Debido a la facilidad de entender lo que es una onda, un sonido y una nota, y viendo el potencial hoy en dia de las redes sociales como Facebook, Twitter o Instagram, han aparecido varios proyectos similares de 'red social para cantantes'. Uno de los más populares es Smule. Smule es una aplicación que permite grabarse, al mismo tiempo ver las letras, y compartir la grabación. Desde hace poco se ha añadido una funcionalidad que también permite detectar el tono y compararlo con una secuencia de notas de referencia pregrabadas.

Otro proyecto análogo: SingSharp. Mucho más restrictivo que Smule, ofrece sin embargo una pantalla que permite seguir el tono de voz de manera gráfica y sencilla.

Esos proyectos muestran el interés que la gente tiene para el canto y la posibilidad de implementar detectores de tono en tiempo real.

1.2.3. Diferenciación del proyecto

A la vista de lo anterior, uno se podría preguntar por el interés real de este proyecto, si ya existen cosas similares. Pues es cierto que la detección de tono, obtener la grabación y publicarla en una red social es un concepto que no es nuevo.

Aquí están ejemplos de fuentes de diferenciación de este proyecto:

- Experiencia usuario: sencillez, rapidez, disponibilidad, velocidad
- El pitch tracker y su implementación propia
- El control concreto sobre el pitch tracker y algunos parámetros internos
- El hecho de que se puede grabar y obtener análisis de manera libre sobre cualquier cosa, con música o no, en base de datos o no
- Las visualizaciones que se ofrecen sobre la afinación, coherencia rítmica...
- Funcionalidades adicionales: como un creador de fichero MIDI y, por tanto, de la partitura (no totalmente implementado a día de hoy)
- Funcionamiento de la red social: nivel de rapidez del servidor...
- El soporte de varios idiomas
- La estética general...

Esta aplicación no pretende generar un tráfico tan denso como lo podría hacer una aplicación profesional como Smule funcionando desde años. Pero sí que podría ofrecer otro enfoque, quizás más pedagógico dando una visión más física y pragmática del canto por ser una realización única, igual que los videojuegos, que entre ellos pueden parecer muy similares, pero que no son competidores directos (es posible jugar a varios).

1.3. Requisitos y tipos de problemas

1.3.1. Experiencia usuario (UX)

Está claro que para que este proyecto sea un éxito, hace falta valorar mucho la experiencia usuario sobre todo a nivel de sencillez de uso, rapidez y estabilidad. La estética es también algo muy importante.

Se tomaron varias decisiones de diseño en este sentido. Entre ellas se pueden destacar la elección de un estilo gráfico homogéneo basado en 2 colores, con un diseño calificado de flat, o el hecho que cada funcionalidad de la aplicación puede ser alcanzada en tan solo 2 clics.

Se dedicó una atención muy intensa al aspecto de la respuesta (responsive) de la aplicación y de su velocidad por una separación en varios hilos de ejecución que estarán detallados en el capítulo 5, que garantiza una fluidez de uso.

1.3.2. Disponibilidad, fiabilidad

Crear una red social y una aplicación que se conecta de manera frecuente a internet sin fallos no es trivial. La disponibilidad debe ser tan alta como sea posible, condicionada en gran medida por la red pero sobre todo por el servidor y la gestión de las peticiones entrantes. Eso plantea problemas de arquitectura, gestión de conexión concurrentes, protocolos de intercambios de datos, etc.

Las problemáticas planteadas requieren del uso de muchos patrones de diseño usados en infinidad de proyectos a gran escala.

Asegurar los intercambios de datos, que sean fiables, exactos a nivel de datos, protegidos para que solo el destinatario acceda a los datos, es otro enfoque que necesita muchas precauciones (por ejemplo, evitar que los usuarios sean capaces de ver datos que no les pertenezca).

1.3.3. Rendimiento y tiempos de ejecución

Debido al hecho de que el proyecto se trata por una parte de una aplicación móvil y por otra parte de un servidor remoto, aparecieron varios requisitos y limitaciones adicionales.

En efecto, hay una gran cantidad de teléfonos Android, con tamaños de pantalla y características muy diversas. Hay que asegurarse de que la aplicación necesite una cantidad de recursos razonable para evitar fallos en otros móviles.Por ejemplo, en Android: si el thread UI principal está ocupado consecutivamente durante más de 30 segundos sin ser capaz de ejecutar sus handlers para refrescar o mandar eventos al sistema, el sistema operativo va a intentar cerrar la aplicación y considerarla caída.

A nivel de servidor: el mayor problema está a nivel de las conexiones entrantes. Tienen que ser tan limitadas como sea posible, para evitar una especie de DoS (Denial of Service) espontáneo, que se suele traducir por una devolución de un código de respuesta 503 (Service Unavailable) hasta que se tomen medidas concretas para

levantar el servidor. No sólo el número de conexiones pero también el tiempo de procesamiento de cada una de ellas debe ser restringido. Eso implica una necesidad de ejecutar cuantas menos instrucciones posibles, y está necesidad se refleja mucho en las tecnologías usadas y la implementación, que se presentarán en los capítulos 3 y 4.

1.4. Metodología

1.4.1. La gestión del tiempo y las etapas importantes del proyecto

Debido a la diversidad de los problemas, fue imprescindible implementar una buena gestión del tiempo y de las prioridades para poder presentar un prototipo funcional. El proceso que se siguió es el siguiente: para cada funcionalidad que se quiere implementar:

- Hacer un listado de los objetivos concretos
- Hacer un análisis de cada uno de los objetivos: requisitos y problemas, el tiempo que se puede dedicar a ellos. Si hay subproblemas, objetivos o sub- funcionalidades emergentes, empezar de nuevo desde la etapa anterior para cada uno de ellos.
- Separar en tareas concretas, generalmente cortas de unos 30 minutos, 1 hora como máximo. Hacer una estimación de tiempo de cada tarea y ver si hay un orden natural o necesario entre las diferentes tareas.
- Hacer búsquedas para conjuntar todo el conocimiento necesario para la buena realización. Eso se repetirá tantas veces como sea necesario durante la realización de dichas tareas.
- Fase de diseño de solución: intentar medir los pros y los contras de cada solución factible que surge, y preparar la organización del código y de los objetos, secuencias de acciones a ejecutar.
- Fase de implementación: redactar el código, muchas veces empezando por pseudo código escrito en etapas. Separar cada etapa en una función. Si dicha función se aplica sobre un conjunto de variable muy peculiar (datos de una nota por ejemplo), crear un objeto que encapsule esta lógica. Reorganizar las carpetas si necesario y hacer tantos commits gracias a un sistema de versioning (git) como haga falta para garantizar la posibilidad de volver atrás.
- Fase de prueba: asegurarse de que los requisitos están superados. Si hay bugs, largos de corregir e investigar, apuntarlos para añadirlos a la lista de tareas después de las tareas más prioritarias.

La figura 1.1 a continuación permite ver las tareas seguidas y las semanas dedicadas a ellas. Es aproximativo porque las tareas una vez acabadas entran en la fase de pruebas y mejora continua.

1.4.2. Las fuentes de información y de ayuda

Algo está claro: no se puede ser experto en todas las tecnologías hoy en día. Sobre todo en un proyecto como éste que usa las tecnologías siguientes: Java, Android, Php, Nginx, las bases de datos MySQL y SQLite, los lenguajes de páginas

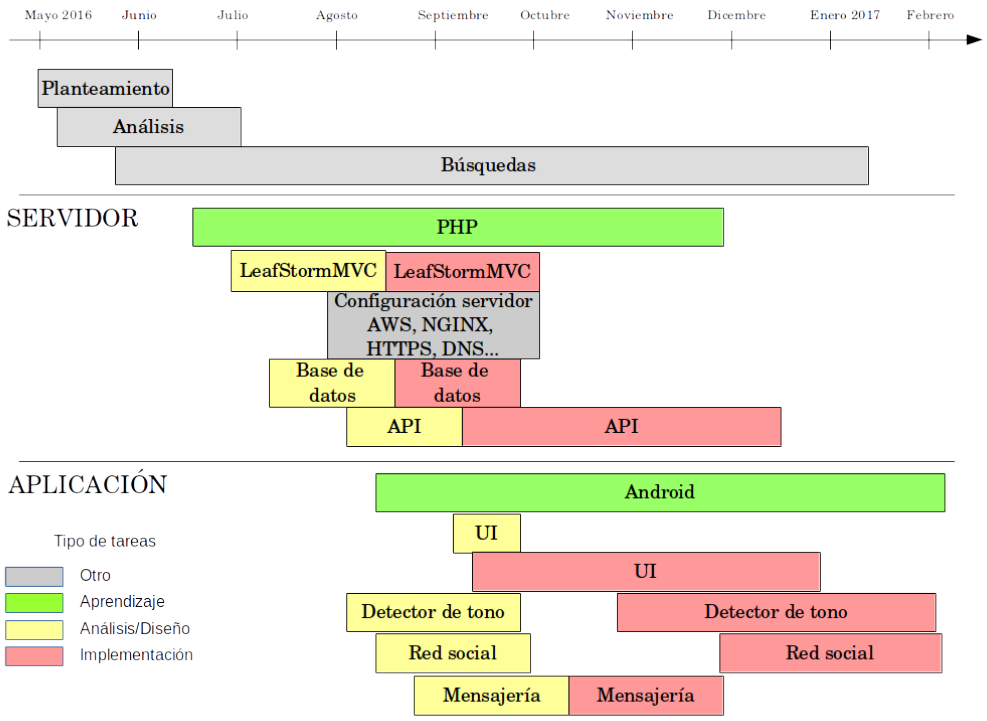


FIGURA 1.1: Cronología de las tareas realizadas

web (HTML, CSS, Javascript), los sistemas de cache como Redis, de mensajería (RabbitMQ...), los servicios de Cloud de Amazon, las tecnologías de criptografía de comunicaciones HTTPS...

No solo sobre las tecnologías. Elaborar un pitch tracker y una aplicación con objetivos de esta magnitud necesita muchos conocimientos matemáticos y físicos, y una cierta experiencia de lo que se puede hacer o no, sobre todo, en tiempo real. Así que unas de las claves de la realización de este proyecto fue la búsqueda constante de información, un problema cada vez, poco a poco, mediante internet y foros de todo tipo, documentaciones oficiales, y muy importante: las preguntas y consultas a investigadores o a ingenieros experimentados y personas cualificadas (referirse a la página de Agradecimientos).

Tampoco se deben olvidar los usuarios finales y probar la aplicación con varios de ellos fue una gran ayuda para retocar diseños y funcionalidades.

Elaborar este trabajo ha consistido entonces, sobre todo, en buscar y analizar información, más que utilizarlas y producir código, lo que justifica la cronología mostrada antes.

1.4.3. La producción de código modular y fácil de entender y manipular

Debido a la variedad de los objetivos y problemas que hay que enfrentar para realizar una aplicación Android y un servidor pensado para una red social, se necesita una organización muy metódica del código.

Es importante remarcar que algo que salvó varias veces la progresión de este proyecto ha sido el sistema de versioning: git, en este caso. Este proyecto tiene dos repositorios git diferentes: uno para el servidor, otro para el cliente, y antes de hacer cualquier cambio importante o después de haber implementado una funcionalidad

difícil siempre suele haber un commit para poder volver atrás si aparecen después regresiones sin razones aparentes (y con tecnologías tan abiertas y 'caprichosas' como puede ser Android, eso ocurrió unas cuantas veces).

Otro punto clave. La organización en carpetas y las abstracciones. Un punto que estará más detallado en los apartados siguientes. Estar acostumbrado a la programación orientada a objetos y las patrones de diseño comunes como Factory, Observer, FactoryMethod, frontend controller, y mucho otros, es una ayuda inestimable. En el código que se presentará y entregará, todo es objeto. No hay código o script sueltos, excepto los de creación de la base de datos en sql. La UI siempre está separada de la lógica, el acceso a la base de datos siempre se hace mediante modelos y una DBInterface, lado servidor o lado cliente. La modularidad del código condiciona su facilidad de entenderlo y de extenderlo y se dedicó mucho tiempo a construir árboles de clases y paquetes coherentes. Ver los anexos A y B y las secciones siguientes para más información.

1.5. Presentación corta de las secciones y anexos

El resto de este documento se ordena siguiendo unos de los planos generales para abordar un problema o una funcionalidad en ingeniería: una parte orientada a diseño (incluye una análisis teórica y práctica) en las secciones 2 y 3, y otra sobre los aspectos más técnicos, la implementación, en las secciones 4 y 5. Se alternan las partes tratando del servidor, y las tratando de la aplicación. Una reflexión sobre el proyecto, sus puntos fuertes y áreas de mejoras, y posible plan de explotación, se encuentran en la ultima sección (sección 6).

Los anexos A hasta F contienen varios diagramas y comentarios acerca de los aspectos más técnicos de la memoria. El código completo desarrollado, se puede encontrar en el Anexo G mediante los repositorios Github ajuntados.

Por fín, se accede al servidor y a informaciones oficiales a destinación de los usuarios o interesados mediante el nombre de dominio 'singvibes.com'.

2 Diseño de la detección de la afinación del canto en tiempo real

2.1. Conceptos

2.1.1. Sonido y ondas sonoras

La música es una suma de sonidos y un sonido es una vibración, la mayoría de las veces, del aire (puede haber sonido, por ejemplo, en el agua). Un sonido, en términos físicos, es una onda progresiva longitudinal la cual se propaga con vibraciones consecutivas de las moléculas que constituyen el medio de propagación. [1]

Un sonido puede ser representado con su forma de onda. Los sonidos son aditivos, es decir, dos sonidos sumados se comportan como un mismo sonido a nivel de onda. Nota interesante: para reproducir un sonido hay que reproducir su forma de onda y eso se hace haciendo vibrar la membrana de un altavoz siguiendo exactamente esta forma de onda. Eso es muy importante para entender el concepto de grabación, que se hace mediante la vibración de una membrana que se encarga de transmitir la amplitud de un forma de onda a un procesador.

2.1.2. Notas y armónicos

Una nota es un sonido producido por un instrumento de música. Está caracterizada por varios parámetros:

- Su frecuencia (fundamental): en Hz, cuanto más alta más aguda
- Su intensidad: o amplitud
- Su duración
- Su timbre: lo que permite diferenciar el sonido de un piano de una flauta, por ejemplo.

La intensidad y la duración son dos propiedades fáciles de entender. La amplitud se suele medir en dBs y cuando el oído humano percibe una variación que le parece lineal de intensidad, la variación real de la intensidad de sonido es exponencial.

La frecuencia y el timbre son dos características que requieren una atención especial para la problemática del análisis de un tono. Para entenderlo bien, hay que recordar una propiedad muy interesante que se aplica a cada señal continua, periódica o pseudo-periódica: la descomposición en series de Fourier.

Una señal continua pseudo-periódica de frecuencia f (llamado fundamental) se puede escribir como una serie de Fourier

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n * cos(2\pi nft) + b_n * sin(2\pi nft)$$

[1] Nociones de física ondulatoria explicadas en el libro *Physique Tout-en-un MPSI-PTSI*, [3]

Con

$$a_n = f \int_{x=0}^{1/f} f(x) * cos(2\pi n f x)$$

$$b_n = f \int_{x=0}^{1/f} f(x) * sin(2\pi n f x)$$

Cuando se toca un La, por ejemplo el La4 de frecuencia fundamental 440Hz al piano, el sonido generado no es un sonido perfectamente sinusoidal. En realidad aparece una sinusoidal de frecuencia 440Hz y sus armónicos (sinusoidales de frecuencias múltiples de la frecuencia fundamental, cuyas amplitudes decaen normalmente hasta 0 con el aumento del valor del armónico).

La transformación de Fourier permite obtener la intensidad de cada armónico (intensidad = f(frecuencia)) en vez de intensidad como función del tiempo. La curva intensidad/frecuencia se llama espectro.

Determinar el tono de una nota, al fin y al cabo, es buscar la frecuencia fundamental de esa nota. ¡Ojo! hay ejemplos de sonidos en los cuales la frecuencia fundamental no está pero todos sus armónicos están (este caso se conoce como frecuencia fundamental aparente) y también es necesario identificarla.

Los problemas, como se puede imaginar, surgen cuando hay varias notas al mismo tiempo y hay que determinar qué armónico o qué componente del espectro pertenece a cada nota. A día de hoy, este problema todavía no se ha resuelto. El resto de este documento hará referencia al canto monofónico (una nota a la vez), obviando así este problema.

2.1.3. Escala cromática y afinación

Este aparatado va a introducir el concepto de afinación y una métrica concreta para estimarla y devolverla al cantante.

En música, existen varias escalas para ordenar las notas dependiendo de su frecuencia fundamental. Una de las más conocidas, y la que se utilizará aquí, es la escala cromática. Es la escala que se utiliza en un piano. Doce notas por octava, y para subir de octava hay que multiplicar la frecuencia fundamental por dos.

Eso se puede explicar con la fórmula siguiente, siendo i el índice de la nota (en el teclado por ejemplo):

$$f_{i+12} = 2fi$$

En efecto, la distribución de las notas de música no es lineal. Considerando una nota de índice i: i siendo, de manera práctica y visual, su número de tecla de piano por ejemplo. Para obtener la frecuencia fundamental de la próxima nota en la escala cromática (fi+1) se hace mediante el cálculo siguiente:

$$f_{i+1} = 2^{\frac{1}{12}} f_i \tag{2.1}$$

Siendo una fórmula definida por recurrencia y una secuencia geométrica, necesita una inicialización. Está inicialización se hace de manera práctica eligiendo el La4, generalmente tomado alrededor de 440Hz.

El origen y justificación matemático/físico/cognitivo de esto no es el objetivo de este trabajo, pero toma su base de los trabajos de Pitágoras y está condicionado por el funcionamiento del oído humano.

A partir de aquí se puede crear una métrica que permite evaluar si el cantante canta afinado o no. Cantar afinado significa emitir sonidos justo a la frecuencia fundamental de una de las notas de la escala musical elegida, aquí cromática. Es mucho

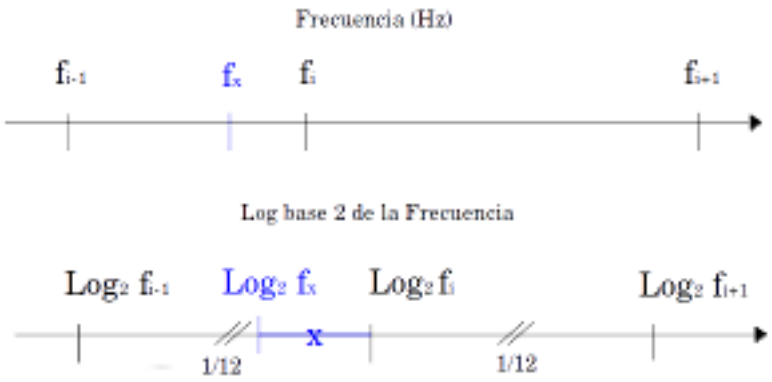


FIGURA 2.1: Definición visual de la afinación

más difícil de lo que puede parecer y sin un oído entrenado y una constante escucha de música o referencias, hay un alto riesgo de trasladar todas las notas hacía arriba o abajo.

El problema es cuantificar está afinación. La idea básica usada es la siguiente: cuanto más cerca de una frecuencia fundamental mejor. Esto introduce el concepto de distancia. La afinación depende de la frecuencia emitida fx, y de la escala musical elegida porque condiciona la nota más cercana.

Sea fx la frecuencia fundamental de la nota emitida, su tono. La primera cosa que habrá que hacer es encontrar cual es la nota más cercana, de índice i.

La afinación (a) puede ser definida de manera sencilla gracias a la figura 2.1 y se basa sobre una propiedad muy importante inducida por la escala cromática: la distancia entre los logaritmos de cualquier frecuencia fundamental de una nota de la escala cromática es constante y vale $1/12$

Demostración: el punto de partida es la ecuación (1)

$$f_{i+1} = 2^{\frac{1}{12}} f_i$$

$$log_2(f_{i+1}) = log_2(2^{\frac{1}{12}} f_i)$$

$$log_2(f_{i+1}) - log_2(f_i) = \frac{1}{12}$$

De manera intuitiva se puede definir la afinación (a) de una nota cantada fx así:

$$a = \frac{log_2(f_x) - log_2(f_i)}{distanciamax}$$

La distancia máxima de log2(fx) hasta log2(fi) es de $1/24$ (por definición de i siendo el índice de la nota más cercana):

$$a = 24.log_2(\frac{f_x}{f_i})$$

Para obtener un valor en porcentaje más cómodo para trabajar, y para poder almacenar este valor como un entero, entre -100 (la nota es demasiado baja) y 100 (la

nota es demasiado aguda), se multiplica este valor por 100:

$$a = 2400.log_2(\frac{f_x}{f_i})$$

Todo el problema ahora reside en la detección de está frecuencia fundamental a partir del sonido grabado por el micrófono, y en tiempo real.

2.2. La detección del tono

2.2.1. Análisis temporal, análisis frecuencial y detección de tono en tiempo real

La detección del tono consiste en la identificación de la nota que se está cantando. Como he dicho antes, una nota no es una sinusoidal, sino una suma de sinusoidales que corresponden a la fundamental y sus armónicos. Hay que tener cuidado, sin embargo, de que la fundamental no siempre está pero la percepción humana hace que la percibamos por sus armónicos (los mismos que la descomposición en series de Fourier introducida anteriormente). Por tanto, el problema consiste, de manera periódica y en una ventana de tiempo corta, en sacar una frecuencia particular que va a corresponder con la nota cantada. Afortunadamente, esta fundamental suele estar definida en las frecuencias más baja de la señal.

Está señal estará, pase lo que pase, dividida en pequeños trozos de una potencia de dos. 1024 o 2048 para una señal de 44100Hz suele ser lo habitual para obtener un tiempo de resolución de 25ms y 50ms respectivamente. Un sample es un valor dado por el micrófono, que corresponde a la amplitud de onda durante un instante determinado (recordamos que un sonido es una vibración del aire, y que una suma de sonidos también). El micrófono devuelve una secuencia de samples que, ensambladas, permiten dibujar la forma de onda de lo que está grabado.

Para obtener mas frecuencias de un bloque de samples, o identificar un patrón, hay dos tipos de métodos. Los métodos basados sobre un análisis en el dominio temporal, o un análisis que basado en una transformación que permite pasar en el dominio frecuencial y obtener una curva amplitud/frecuencia, generalmente la de Fourier (usando la Fast Fourier Transform).

Cada uno de esos métodos tienen ventajas e inconvenientes. Las transformaciones hasta el dominio frecuencial son generalmente mucho más potentes y permiten resultados muchos mejores, pero son mucho más lentas y difíciles de implementar, y usan muchas funciones trigonométricas, que resultan muy lentas de computar incluso usando tablas precalculadas y las series de Taylor-MacLaurin.

Para este proyecto, se ha considerado como lo más adecuado un análisis de la señal poco transformada.

2.2.2. El Fast-Lifting-Wavelet-Transform (FLWT)

Debido al hecho que la frecuencia fundamental de una nota suele ser su frecuencia más baja, o, una división por dos de su segundo armónico si la fundamental no está (los armónicos tienen sus frecuencias que son múltiples de la fundamental), una idea espontánea que surge es la simplificación de la señal aplicando un low-pass filter. Eso descarta informaciones de la señal que no importan y permiten una periodicidad que puede no ser obvia en la señal original (en realidad, las notas son

señales pseudo periódicas amortiguadas). Una transformación particularmente sencilla a aplicar y rápida, puesto que no necesita ninguna clase de cómputo adicional además del propio de sumas, multiplicaciones y divisiones por dos, es el uso del Fast Lifting Wavelet Transform Algorithm. Las Wavelets son señales sencillas diseñadas para realizar comparaciones (convoluciones) con una señal y producir métricas de similitudes y así identificar la presencia o no presencia de un patrón (en su principio más básico). Se trata de un algortimp muy usado para simplificar datos y, por lo tanto, en compresión, al igual que el JPEG2000.

El algoritmo Fast Lifting Transform Wavelet, abreviado en FLTW, consiste en escribir la señal como una suma de dos curvas: una aproximación, y un detalle, tomando una muestra de cada dos, para simplificar la señal tal y como se ha dicho antes. Se suele repetir la operación sobre la parte aproximada hasta obtener una señal muy limpia que contenga solo un par de armónicos de baja frecuencia.

2.2.3. El algoritmo FLWT basado en las Wavelets de Haar

Para obtener una frecuencia determinada, en muchas ocasiones la más baja de la señal, aplicar el algoritmo FLWT construido a partir de las wavelets de Haar es muy eficiente. En efecto, matemáticos como Daubechies y Sweldens ya demostraron que las Wavelets de Haar permiten llegar a la definición de la aproximación y del detalle sacados con las ecuaciones siguientes:

$$a(n) = \frac{x(2n) + x(2n+1)}{2}$$

$$d(n) = x(2n+1) - x(2n)$$

Para encontrar la demostración y la explicación del FLTW con las wavelets de Haar, referirse a los trabajos siguientes: [2]: *Factoring Wavelet Transforms into Lifting Steps*

n es el número de sample dentro del bloque de samples elegido, x la señal, a la aproximación y d el detalle (que se descarta aquí). Esas ecuaciones dependen del tipo de Wavelet usado y no se va a hacer la demostración aquí. Lo importante es que los resultados pueden ser manipulados como enteros y son sencillas. Dos características fundamentales para la resolución en tiempo real.

Aquí están las etapas del algoritmo:

- A partir de los samples de la ventana elegida, crear la aproximación a (que contiene dos veces menos samples y que se comporta como un low pass filter).
- Calcular las variaciones extremas de amplitud.
- Contar los máximos, mínimos, detectar dónde están, para ver si se destaca una frecuencia particular.
- Si no se detecta una frecuencia cuya convolución a la señal no da un resultado bueno, empezar de nuevo. Si ya está alcanzado el número máximo de iteraciones, parar y concluir sobre el hecho que no se puede detectar tono aquí, porqué no hay nota.

Este trabajo, y este algoritmo, se basa sobre el trabajo siguiente: [1]: *Real-Time Time-Domain Pitch Tracking Using Wavelets*. Para mejorar el algoritmo y su fiabilidad, se ha integrado un procesamiento dinámico que consiste en comparar el valor detectado con el anterior. Si hay una variación enorme en muy poco tiempo, probablemente los valores sean erróneos y hay que descartarlos o aplicarles un valor de confianza más débil. En efecto, en el canto, la voz humana, por muy entrenada

que sea, no se puede mover de los bajos a los agudos de manera instantánea. Se encontrará el código del detector de tono en el anexo C.

Este algoritmo es muy eficiente y consume poca memoria, únicamente manipula enteros, por lo que minimiza el impacto de usar un lenguaje de tan alto nivel como Java.

3 Diseño de la red social y elaboración de una arquitectura distribuida

3.1. El paradigma cliente-servidor

3.1.1. Presentación de los actores y de la topología del sistema

El sistema utiliza el paradigma cliente-servidor. Consiste en la implementación de un servidor capaz de intercambiar datos con uno o varios clientes. Eso supone que el servidor es capaz de escuchar a las conexiones entrantes, decodificar la petición, procesarla, a veces almacenar datos y ejecutar computaciones dependiendo de esta misma petición, construir una respuesta y devolverla al cliente que mandó la petición.

En nuestro caso, la aplicación móvil tiene el papel de cliente, conectado al servidor mediante internet (red celular, o wifi generalmente). El servidor está localizado en una instancia EC2 (elastic-cloud) de Amazon. Nginx permite hacer la interfaz entre los ficheros del servidor y el trafico entrante/saliendo mediante el protocolo HTTPS. El uso de este protocolo está implementado mediante un protocolo SSL obtenido por un proveedor de certificado automático llamado Let's Encrypt. Más información se puede encontrar en el enlace de la bibliografía [11].

Un esquema de la estructura básica del sistema se encuentra en la figura 3.1.

3.1.2. Tipos de datos y base de datos

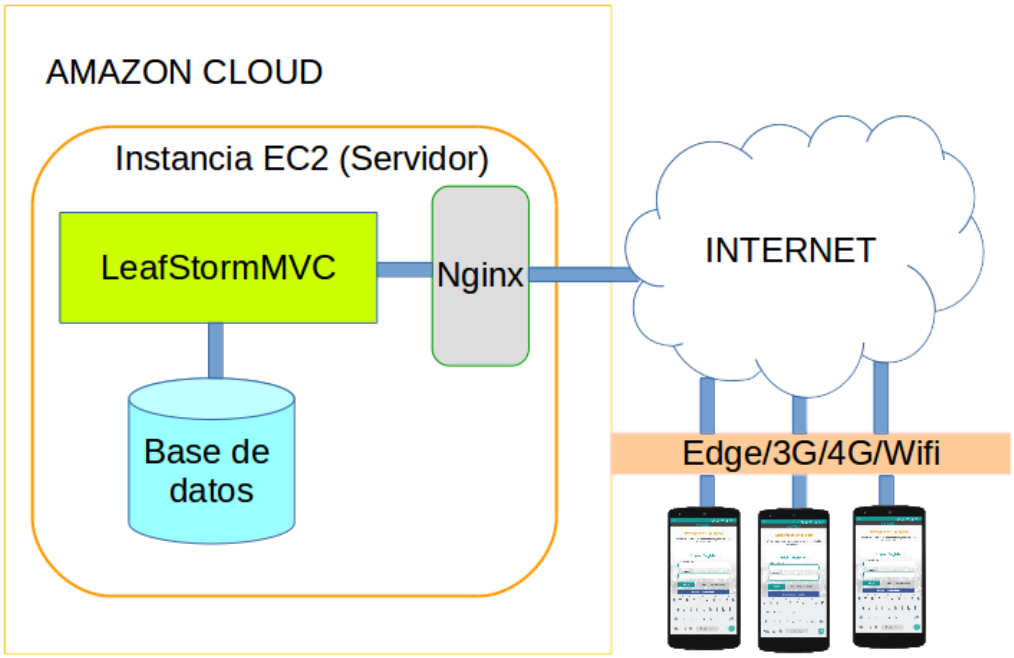
Los datos son de varios tipos.

- Datos de usuarios: datos personales como el nombre, correo electrónico, 'digest' de la contraseña, seguidores, mensajes privados...
- Datos de las grabaciones: ficheros audio, resultados de la análisis por el detector de tono, canción asociada
- Datos sobre las canciones: artistas, títulos, letras
- Imágenes: las de perfil de usuarios por ejemplo

Para realizar una base de datos fácil de usar sin redundancia entre los datos, se aplicó un esquema cumpliendo los requisitos de la forma normal de Boyce Codd (una clave primaria por tabla, no dependencias funcionales entre los atributos no claves).

Además, se realizó un esquema similar a un esquema en estrella cuyo centro, o tabla de hechos, son los Recordings (grabaciones). Las grabaciones tienen varias dimensiones: una dimensión para los usuarios, una para las canciones, una para el tiempo. Se plantea hacer un análisis de los datos para mejorar la aplicación, destacar

FIGURA 3.1: Arquitectura del sistema



ciertos perfiles de usuarios, las canciones más populares, para al final llevar a cabo estudios y producir informes como se haría en un contexto de Big Data con todas las problemáticas de Data Warehouse asociadas (con un proceso ETL sencillo por tener los datos generados por la aplicación, fuente única, datos almacenados de manera continua en la base de datos).

El diagrama UML representando el esquema de la base de datos se puede encontrar en el anexo F. Tal como está ahora, es bastante cómodo a nivel operativo, los controladores nunca necesitan más de un join para devolver los datos necesarios para la aplicación.

La gestión de los datos de sesión requiere una atención muy peculiar, gestionada mediante la tabla Tokens, cuya justificación y diseño será explicado en la sección siguiente y permite implementar un cierto control de acceso sobre los datos conjuntamente a un cierto diseño de los controladores de los websevicios, introducidos en la parte siguiente.

3.1.3. Intercambios de datos, mensajes y protocolos

Los datos son de varios tipos y generados mayoritariamente desde la aplicación cliente. La aplicación cliente dispone de clases permitiendo el uso del protocolo HTTPS (petición HTTP cuyo contenido está criptado mediante el protocolo SSL, cuyo funcionamiento y decodificación se puede hacer mediante un certificado).

El servidor devuelve sus respuestas a la aplicación cliente en JSON, formato muy conciso para representar objetos, que son el resultado del patrón de diseño DAO, Data Access Object extiendo la clase ModelAbstract (ver la parte sobre el Modelo Vista Controlador en la parte 4).

También puede devolver ocasionalmente binario a la aplicación cliente, cuando se solicita un fichero audio o una imagen.

El servidor, usado mediante un navegador con urls cuyo frontname es diferente de 'api', en lugar de devuelve JSON en texto plano devuelve html.

Aquí está una lista de las rutas de la API implementadas, parámetros necesarios, y tipos de repuesta:

Ruta	Paramétros necesarios	Variables en el JSON si éxito	Resumen
/login	email, password	success, token, message	Devuelve un token de sesión
/logout	token	success	Destruye el token de sesión y marca el usuario como desconectado
/register	email, password, username	success, message	Crea un nuevo usuario
/account/get	token, user_id	success, user	Devuelve datos del usuario user_id
/account /edit	token, user_data	success	Edita los datos de usuario aplicando los campos contenidos en la tabla user_data
/account /follow	token, user_id	success	Hace que el usuario siga al usuario con el id: user_id
/account /unfollow	token, user_id	success	Usada para parar de seguir a un usuario
/account /followedby	token, user_id, last_updated_at	success, users=[], followers=[]	Obtiene la lista de los seguidores (los users) del usuario con id 'user_id', y devuelve las relaciones Followers
/account /following	token, user_id, last_updated_at	success, users=[], followers=[]	Lo mismo, pero esta vez para obtener los usuarios que están seguidos por el user con id user_id
/account /search	token, needle	success, users	devuelve una collección de usuarios cuyo username, firstname, lastname concatenados contienen la cadena 'needle'
/messages/get	token, last_updated_at email	success, messages	Obtiene los mensajes recibidos a partir de 'last_updated_at' hasta ahora del usuario
/messages /send	token, user_id, content	success	Manda un mensaje ('content') al usuario con id user_id
/news /newsfeed	token, last_updated_at	success, multilocales, announcements, newer_recordings, followed_people_recordings	Recupera un conjunto de datos utilizado para construir el newsfeed en la aplicación a partir de las grabaciones...
/news/popular	token	success, popular_songs	Devuelve las canciones más grabadas por los miembros
/news /notifications	token, last_updated_at	success, follow_queries, comments_on_my_recordings, new_likes	Devuelve lis nuevos comentarios y nuevos seguidores
/recording /get	token, last_updated_at, recording_id	success, recording	Obtiene los datos del Recording cuyo id es recording_id
/recording /getby	token, last_updated_at, user_id	success, recordings	Recordings contiene todos los recordings públicos hechos por el usuario identificado por user_id

/recording /edit	token, last_updated_at, recording_id, recording_data	success	Permite editar los datos de un recording
/recording /upload	token, recording, audio	success	Almacena una nueva grabación en el servidor, haciéndola pública
/recording /loadcomments	token, last_updated_at, recording_id	success, comments	Devuelve los comentarios de una grabación
/recording /comment	token, last_updated_at, recording_id, comment	success	Deja un comentario 'comment' en el recording identificado por 'recording_id'

CUADRO 3.1: Tabla de las rutas de la API: parámetros, respuesta y descripción

Hecho a notar: todas las peticiones se hacen con el verbo HTTP POST, porque pasar el token de sesión es necesario para cualquier petición de la API excepto la de login y de register. Pasarlo por GET, (por la URL) es peligroso y no recomendable porque este token debe ser solo conocido por el servidor y la aplicación cliente, y los parámetros en la URL se pueden recuperar con facilidad.

3.2. Funcionalidades y problemas mayores en el diseño y la implementación de la red social

3.2.1. Seguridad y sesiones

Una red social se basa sobre un conjunto de usuarios, que deben poder acceder al mismo tiempo de forma concurrente a la aplicación y que, dependiendo de su identidad, no tenga acceso a los mismos datos que otro usuario.

Entonces el primer problema a resolver es la identidad y el proceso de autenticación (reconocimiento por el servidor de está identidad): Singvibes identifica un usuario gracias a su correo electrónico durante la autenticación, le devuelve un token de sesión, almacenado en base de datos juntamente al id del usuario en la tabla Tokens y una fecha de expiración (30 minutos después de ser emitido). Este token sirve de sustitución a los credenciales hasta que expire, para ser sustituido por otro. (referirse a las clases SessionManager.php y TokenManager.php del servidor, en raiz/base/security)

Los controladores de API, salvo el Index (que permite hacer el login) siempre comprueban la existencia del token en los parámetros mandados por la aplicación cliente en los parámetros POST de las peticiones HTTPS antes de devolver una respuesta para garantizar que solo miembros de Singvibes inscritos puedan recuperar los datos.

Sin embargo, el uso de redes inseguras inalámbricas hace posibles ataques de tipo Man In The Middle entre otras cosas, o replay. El libro [4]: *Hacking for Dummies* provee varias informaciones sobre estos tipos de amenazas.

En la tabla 3.1 se presenta una lista de algunos fallos de seguridad usuales y las medidas tomadas.

Problema	Soluciones implementadas
Las redes usadas por un telefóno móvil no son seguras: puede ser bastante fácil de leer/interceptar paquetes y hacer un ataque de tipo Man Of The Middle	Uso de un criptaje de cada petición HTTP con SSL (HTTPS). También hay tokens de sesión generados por el servidor, necesarios para utilizar la API. Se obtiene gracias al login, y el hecho que expire frecuentemente ayuda a mitigar este tipo de ataque.
Cross-scripting (XSS)	Todo el código SQL está autogenerado con valores escapados. No se puede entonces (en teoría, que hasta las empresas profesionales como Wordpress o Magento tienen o tuvieron fallos de este tipo) almacenar código php en la base de datos que entonces podría ejecutarse cuando se imprimen dichos datos.
Acceso a ficheros del servidor no deseados y ejecución remota de código	El servidor Nginx está configurado para solo devolver los assets estáticos. Las carpetas escondidas están bloqueadas con una redirección sistemática a 403 Forbidden. El resto de las peticiones entra en el index.php, y el Router no sabe hacer nada más que ejecutar código de los controladores programados.
Ataques Replay, repetición	este tipo de ataque es problemático por ahora. Consiste en la estricta repetición de una petición. Es peligroso porque no necesita el conocimiento del contenido de la petición o sus parámetros para funcionar. Los tokens de sesión tienen una duración de vida limitada. También, muchas peticiones requieren un campo 'last_updated_at' que impide obtener datos más antiguos que está fecha. Una solución mucho más segura sería combinar el token de sesión con un token rotativo válido una sola vez.
Ataques de tipo DDOS (Denial of Service, saturación de un servidor usando generalmente muchos ordenadores o robots al mismo tiempo)	Por desgracia no hay mucho que se puede hacer, sino tener otro servidor por delante que recibe las peticiones y que si se satura por culpa de un DDOS que al menos el 'verdadero' servidor no sufre las consecuencias. Un DDOS sin embargo es costoso de hacer, y por ahora no sería rentable con un proyecto estudiantil como este. De hecho, los DDOS suelen ser cortos puesto que en general cuesta mucho.

CUADRO 3.2: Amenazas, ejemplos de fallos de seguridad usuales, y soluciones encontradas

3.2.2. Comunicación entre usuarios y puesta en contacto

Una red social debe permitir a los usuarios encontrar a otros miembros. Un buen ejemplo es Facebook, en el cual se inspiró este proyecto: contiene un newsfeed continuamente alimentado por publicaciones de otros usuarios, una mensajería instantánea, y un buscador de personas (entre otras cosas). Todo esto favorece la puesta en contacto natura de los usuarios entre ellos.

La comunicación entre usuarios debe ser posible mediante el servidor, que almacena mensajes y acciones de los usuarios y las devuelve dependiendo de la identidad del cliente, que puede ser recuperada gracias a las medidas explicadas en el apartado anterior.

La mensajería instantánea necesitó mucha reflexión. No hay presupuesto para aguantar un servidor tan potente como uno de Facebook. La carga de un servidor depende de varios factores pero los más importantes son el número de usuarios, número de conexiones entrantes al segundo y volumen de datos intercambiados en cada una de las peticiones. Por ahora, la solución ha sido un refresco periódico de la ventana de conversación 'instantánea' cada 1 minuto cuando está abierta.

4 El servidor: elaboración e implementación

4.1. El servidor y su organización interna

4.1.1. Presentación del servidor

El servidor tiene la responsabilidad de proveer datos a las aplicaciones clientes instaladas en los móviles de los usuarios y de darles posibilidad de intercambiar entre ellos, (paradigma cliente->servidor), como se ha descrito antes.

Está accesible desde el nombre de dominio: singvibes.com que está mapeado a una IP de una instancia EC2 de Amazon, máquina virtual en el Cloud de Amazon.

Esta instancia tiene el sistema operativo Ubuntu Server, y por ahora aloja toda la base de datos y los ficheros permitiendo el buen funcionamiento de Singvibes. La base de datos es una base de datos relacional MySQL cuyo esquema puede ser encontrado en el anexo F. El código del servidor está entregado con esta memoria. Es una carpeta compuesta mayoritariamente de ficheros PHP, enlazados a Nginx, programa capaz de escuchar a las conexiones entrantes en HTTP(S) y hacer que esta instancia EC2 se comporte como un verdadero servidor invocando los ficheros necesarios para devolver una respuesta al cliente.

Cuando el tamaño de los datos haya crecido, se plantea utilizar un Amazon S3 para almacenar los datos pero con una disponibilidad razonable.

4.1.2. Pros y contras de varios frameworks MVC

Muchos frameworks están siendo utilizados para facilitar la construcción de una aplicación web. Un framework es un conjunto de librerías que permite la estructuración del código de una aplicación mediante uso de varias clases, varios paradigmas.

Uno de los paradigmas más usados en la arquitectura view es el Model-View-Controller. La idea básica es la siguiente: para renderizar una página o una respuesta a una petición, un controlador examina los parámetros de la petición, genera vistas (generalmente templates que contienen código html completado por instrucciones que se ejecutan en el servidor), y para renderizar dichas vistas debe interrogar a modelos, que proveen una interfaz y una manera de manipular datos de una o varias bases de datos. Generalmente hay un modelo por tabla.

Un framework es muchas veces genérico. Lo que es genérico en informática suele ser muy elegante, pero, por implicar lógica de alto nivel y anticipación de muchos casos particulares, son muy pesados. El mejor ejemplo de esto es Magento, y sobre todo Magento 2, framework que permite la creación de tiendas online. Es muy modular y tiene una gran flexibilidad. Sin embargo, este framework pesa mucho, con varios miles de ficheros y unas 400 tablas de base de datos. Está basado sobre Zend (y Symphony para Magento 2), otro framework PHP, un poco más ligero. Otros frameworks como Ruby On Rails parecen ligeros y muy faciles de usar, sin embargo, Ruby, por ser un lenguaje de alto nivel, ya mostró sus limites en varios proyectos en

Ruby on Rails [1]. Spring MVC, basado sobre Java, tiene muchos automatismos, mediante las anotaciones, una 'magia' en el vocabulario informático muy potente que está totalmente en contradicción con uno de los requisitos y principios fundamentales que aplico a la hora de escribir código para un servidor: ejecutar cuantas menos instrucciones posibles. Por tanto, no se va a hacer uso de una máquina virtual Java que puede consumir hasta tres cuartos de los ciclos de CPU, ni lenguaje de alto nivel como Ruby que ejecuta mucho código de C por debajo, aunque tenga una sintaxis muy expresiva. Un compromiso razonable entre tiempo de desarrollo y rendimiento parece ser el conjunto del servidor Nginx con PHP. El problema es que PHP se puede volver dificil de organizar y es muy difícil de usar, por permitir tanta libertad a partir de tantas funciones [2]. Pero, con un cierto control, patrones de diseños adecuados, el establecimiento de routing apropiado y de un autoloader, permite hacer mucho en pocas lineas y pocas instrucciones ejecutadas.

Por eso, como este proyecto no exige mucho a nivel de frontend, o a nivel de operaciones sino únicamente obtener datos, comprobar cosas y devolverlas, pareció más razonable utilizar un pequeño framework MVC casero desarrollado justo a medida para este proyecto, con solo las funcionalidades necesaria. Este micro framework estará presentado brevemente en la sección siguiente.

4.1.3. Diseño: front controller, organización de los ficheros

Una manera muy cómoda, común a todos los frameworks comunes usados en aplicaciones web, de organizar los ficheros y las carpetas del Modelo-Vista-Controlador es usar un paradigma que se llama el FrontEnd Controller. Básicamente consiste en proporcionar a la aplicación un punto de entrada único, en el caso de una aplicación php, un 'index.php'. Posteriormente, un fichero '.htaccess' en el caso de un servidor Apache o un bloque server en el caso de Nginx, redirigirá todas las peticiones hasta una url del subdominio que no sea un 'asset' (css, imagen, javascript...) hasta el 'index.php'.

El 'index.php' se encargará después de invocar un router, capaz de instanciar dependiendo de la url la clase de controlador correcto e iniciar el proceso típico del MVC (controlador que genera vistas interrogando la base de datos a través de modelos).

Si no se hace esto, en php, hay que tener un fichero por url, y en la dirección indicada por la url, lo que impide toda forma de abstracción y de modularidad.

4.2. LeafStormMVC: implementación propia del paradigma MVC

4.2.1. Request flow: routing y controladores

Cuando una petición HTTPS entra en el servidor Nginx, si la petición no es un asset (imagen, css, javascript, etc), la petición está redirigida hasta el fichero 'index.php'. Este fichero es responsable de invocar dos clases: el Autoloader y el Router. El autoloader se encarga de encontrar las clases y cargar los ficheros que las contienen cuando están invocadas. Eso permite evitar la inclusión de todas las clases PHP del framework. El router es una clase especial cuya responsabilidad es la de analizar

[1] Un articulo corto sobre los pros y contras de Ruby on Rails: https://www.madetech.com/blog/pros-and-cons-of-ruby-on-rails

[2] Hay que ver la documentación oficial de PHP y el número de funciones para darse cuenta. http://php.net/manual/en/index.php

la URL y los parámetros para delegar el procesamiento a un controlador adecuado, implementando así el Front Controller pattern, un patrón muy útil para organizar el código y tener un mayor grado de control sobre el código ejecutado. En caso de error, si la url no existe, se devuelve la página 404.

Las URL con LeafStormMVC siguen exactamente el mismo patrón que varios frameworks en los cuales está inspirado (Magento/Zend). Se componen de 4 partes: el nombre de dominio, un frontname, un nombre de controlador y, finalmente, una acción. Puede estar seguida o no de parámetros GET (después de un interrogante).

Por ejemplo: la URL https://singvibes.com/api/account/get

Si falta el frontname, el frontname 'web', se aplicará por defecto. Si el controlador o la acción faltan, el IndexController y la indexAction se aplicarán, devolviendo la home page del servidor. Entonces, esas 4 URLs son equivalentes y devuelven la homepage:

- https://singvibes.com
- https://singvibes.com/web
- https://singvibes.com/web/index
- https://singvibes.com/web/index/index

Eso da mucha flexibilidad, las rutas existentes sólo son definidas por la arquitectura de las carpetas. Así, la generación de un frontname 'api' se hace creando una carpeta en la localización raíz del proyecto /controllers/api. Para crear un Controlador Account se construye una clase llamada AccountController en el fichero proyecto /controllers/api/AccountController extediendo de la clase ControllerAbstract, conteniendo los métodos para gestionar los parámetros, y una función render() que se encarga de construir la respuesta, cuyo comportamiento está definido en las clases derivadas. El framework ya integra dos tipos de controladores:

- ControllerWebAbstract: Los controladores que se extienden de esta clase devuelven código HTML, construido a partir de una inclusión automática de templates .phtml localizados en la carpeta raiz/views/web. También permiten controlar de una manera básica el SEO (Search Engine Optimization, el posicionamiento de un sitio en los motores de búsqueda) con los robots (INDEX FOLLOW...), y en breve, con los datos estructurados tipo JSON-LD .
- ControllerWsAbstract: Esos controladores proveen varios métodos adecuados para devolver datos en formato JSON, y averiguar la autenticidad de un usuario basándose sobre su token de sesión, si está presente en los parámetros POST. JSON se estandardizó poco a poco sustituyendo a otro formato muy popular para intercambiar datos en formato texto: el XML, más verboso.

Para crear un controlador dedicado a la gestión de los datos de usuario devolviendo datos en JSON, se debe:

- Crear la carpeta controllers/api si no existe ya
- Crear el fichero controllers/api/AccountController.php
- Crear la clase AccountController dentro de este fichero y que extienda de ControllerWsAbstract, clase formando parte de la base de LeafStormMVC

- Crear las acciones: cuya forma más básica es public void nombreAction() { $this->render(); }, devolviendo el JSON siguiente {'success':'false'}. Para añadir datos al JSON se usa el método addToResponse($key, $object), que se encarga de, dependiendo el típo de objeto, añade al JSON un campo 'key' conteniendo la representación JSON del objeto $object.

Como he explicado antes, el Router ya es capaz con esto de saber qué nuevas rutas a /api/account/... fueron añadidas y son accesibles. En el caso de la API, cada método también comprueba si el usuario posee un token de sesión que no está expirado.

4.2.2. Los modelos

Para implementar a la vez el paradigma Slim Controller (controladores ligeros), y el Modelo Vista Controlador, se crearon los modelos. Los modelos, en LeafStormMVC como en varios frameworks como Ruby On Rails, extienden de una clase ModelAbstract. Está clase provee todos los métodos necesarios para realizar las CRUD operations (Create, Read, Update, Delete), y eso mediante una clase llamada DBInterface y la QueryFactory.

Manipular SQL sin errores en texto plano es pesado y fuente de muchos errores. La clase QueryFactory proporciona métodos para crear peticiones SQL adaptadas para las operaciones CRUD básicas (select, insert, update, delete), implementando el patrón de diseño llamado Factory en el desarrollo de software.

Un controlador, para renderizar las vistas necesita interactuar con los modelos y cambiar el estado de los datos. Eso se hace de manera sencilla gracias a las clases que extienden de ModelAbstract. En LeafStormMVC como en Ruby on Rails, se suele hacer un modelo por tabla, con los campos accessibles definidos antes. No se puede modificar u obtener los datos de un campo que no está declarado en dicho modelo. Eso asegura la encapsulación de los datos y un control más potente.

Se puede encontrar el código completo de las clases principales del framework en el repositorio Github del servidor (enlaces disponibles en el anexo G, y el código del AccountController que muestra varias aplicaciones de los modelos.

4.2.3. Las vistas: gestión del frontend

El frontend está gestionado mediante un sístema básico de inclusión de ficheros phtml (html conteniendo inclusiones de php), mediante la clase ControllerWebAbstract, que provee la función addBodyTemplate. Cuando la función render() es llamada, los templates son encadenados y evaluados. Pueden estar localizados en cualquier sitio dentro de la carpeta raiz/views/web. El css, javascript y otros assets están en raiz/views/assets.

5 Singvibes para Android

5.1. Estructura de la aplicación y organización del código

5.1.1. Presentación global de la aplicación

Singvibes es una aplicación sencilla, con un diseño flat, y un pequeño conjunto de pestañas/ventanas y muy pocos menús. Las pantallas en Android se llaman Actividades. Hay dos en toda la aplicación. La LoginActivity y la MainActivity. La LoginActivity consiste en un formulario de login/register. Singvibes siendo una red social, está pensada para funcionar en línea. La MainActivity está dividida en 4 fragmentos, que son pestañas separadas, y tiene un menú de opciones tal como el idioma y el login automático. Las pestañas son las siguientes:

- NewsFragment: una pestaña conteniendo las canciones populares (las más grabadas) y el newsfeed, que muestra grabaciones hechas por varios usuarios. Pueden ser seguidores, o no y permite al usuario escuchar otras grabaciones (está implementado pero por una limitación por ahora no está funcionando).
- FriendsFragment: una pestaña que funciona como un mini Whatsapp. Permite encontrar a personas y hablar con ellos mediante un chat. Muestra también la actividad reciente: un nuevo seguidor o un nuevo comentario sobre las grabaciones hechas.
- SingFragment: La parte más importante, la que permite grabarse, y visualizar la curva de afinación. (mostrar imagen aquí) Una vez acaba la grabación, se puede escuchar y ver de nuevo la gráfica. También se puede decidir ponerla en el servidor para que la gente lo escuche.
- HomeFragment: El perfil del usuario de la aplicación. Contiene un formulario que permite al usuario cambiar sus datos, dar informaciones para los otros usuarios o la lista de sus grabaciones hechas previamente. También se puede consultar aquí la lista de los seguidores.

Esta aplicación está pensada para que cada funcionalidad sea disponible en dos clicks o menos. Disponible en Español, Inglés y Francés, con una interfaz minimalista e intuitiva.

La aplicación es capaz de conectarse a la red y hacer peticiones HTTPS para recibir datos desde el servidor que se almacenarán en una base de datos local en SQLite (nativamente soportada por Android, aúnque solo es una solución temporal por ser poco segura y protegida). Todos los procesamientos largos tipo descargas o computaciones se hacen en hilos separados Android, reservando el thread principal para la UI, considerando la aplicación como bloqueada si el hilo no está libre durante más de 30 segundos, lo que además bloquea el usuario.

5.1.2. Organización del código y modularidad

Organizar el código en Android no es cosa fácil debido a la lógica peculiar de gestión de los hilos, de los observadores de eventos y de los listeners (código ejecutado cuando un evento pasa, como cuando se teclea algo). La mayoría de los tutoriales muestran la creación de muchas clases privadas dentro de clases públicas que no son reusables y que hacen el código de las actividades muy pesado, mezclando lógica, UI, listeners, handlers... Porque, casí cualquier cosa que interactúa con la UI o con el framework de Android necesita una referencia a un objeto de la clase Context o Activity, dos objetos que son extremadamente pesados.

Esto obligó a realizar varias reestructuraciones de la carpeta del proyecto hasta acabar con el anexo C.

Un concepto muy importante en las aplicaciones con interfaz gráfica parece ser la separación entre la parte lógica y gráfica, separación completamente antinatural en Android debido al hecho de que el hilo principal es el hilo gráfico, y es único, sólo se pueden ejecutar operaciones de dibujo allí.

La consecuencia fue llevar un diseño por capas. La entrada se hace por las actividades, que instancian varios objetos lógicos y gráficos, y los lógicos comunican sus resultados a los gráficos que se renderizan invocando el hilo gráfico, muchas veces de manera asíncrona porque los procesamientos largos (peticiones HTTPS, detección de tono, preparar los puntos de una gráfica antes de dibujarlos) se hacen en otros hilos.

El acceso a la base de datos se hace mediante las mismas clases que con el framework usado en el servidor, adaptadas para Java y SQLite, y tienen el mismo proposito que en el servidor: es decir no preocuparse de escribir lineas de SQL manualmente, y permiten las líneas de las tablas de base de datos como si fueran objetos, que se pueden instanciar y que dan acceso a sus valores mediante accesores (get, set...). (patrón de diseño DAO - Data Access Object).

5.1.3. Multithreading en Android

Como se ha mencionado antes, en Android, el único hilo que existe al entrar en la aplicación es el hilo gráfico, el principal, el único que permite dibujar, y el que intercepta los eventos, los inputs del usuario. Si se mantiene demasiado ocupado, la aplicación cierra con un error. Entonces, hay que pensar un ciclo de vida para los hilos de ejecución muy robusto, que además sepa gestionar las interrupciones de la aplicación y su vuelta (cuando el telefono se bloquea y desbloquea...).

Una tarea puede ser ejecutada de asíncronamente de varias maneras: mediante un objeto que extiende la clase AsyncTask, Runnable o Thread. Los dos últimos funcionan de manera más o menos igual.

En este momento de desarrollo la aplicación dispone de varios threads creados para realizar varias tareas.

- El thread gráfico: el principal
- Unas tareas asíncronas cada vez que se solicita un acceso a la red para recuperar datos. Actualizan la vista una vez que acaban mediante una función llamada onPostExecute, que siempre se ejecuta en el hilo principal cuando se acaban las tareas especificadas en el método doInBackground.
- Varios threads lógicos, uno para cada tarea costosa: conputación del tono de voz en tiempo real, refresco de los puntos de una gráfica...

5.2. La implementación del pitch tracker

5.2.1. Recuperar la señal

Recuperar la señal de un micrófono en Android se puede hacer de dos maneras. O usando un objeto que se llama MediaRecorder, que automáticamente es capaz de convertir el flujo de datos desde una camara o un microfono (o ambos) en un formato específico, pero no da acceso al buffer de entrada (entonces no hay acceso a las samples), o el AudioRecord. Este objeto es interesante porque alimenta un buffer de samples que se puede leer de manera continua y se rellena a partir del micrófono elegido (algunos teléfonos tienen varios). Sin embargo, es la responsabilidad del desarrollador de después ensamblar las muestras en un fichero si se quiere guardar, y no será un fichero comprimido (será un fichero WAV), excepto si se implementa un algoritmo de compresión, o si se consigue usar una librería externa como ffmpeg para convertir la grabación, cosa difícil de hacer en Android.

Para acceder al micrófono, la aplicación debe tener los permisos declarados en el AndroidManifest.xml, un fichero declarativo conteniendo varias variables de configuración general, y dichos permisos deben ser activados en las opciones de las aplicaciones en los parámetros Android (está activación de permisos se hace automáticamente desde el Google Play cuando se instala la aplicación).

Para utilizar la clase AudioRecord, se creó una clase llamada Recorder, que lleva una referencia hasta una instancia de AudioRecord, y varias constantes útiles como el sample rate, el tamaño de los bloques de samples que se van a mandar al detector de tono, etc. Esta clase Recorder se invoca mediante un botón de grabación, llamando, a su vez, a dos métodos dependiendo del estado de proceso (grabando o no grabando), start y stop, que se encargan de iniciar y cerrar el proceso de grabación. Cuando se llama a start(), un bucle (ejecutado en un thread separado mediante los objetos Runnable de Android) se encarga de leer las muestras añadidas en el buffer del AudioRecord, y de invocar el detector de tono, que devuelve la frecuencia fundamental asociada a este bloque de samples, si existe para esa ventana. Este dato se envía a la Actividad principal mediante la función publishFrequency, ejecutada en el thread principal (el thread de la interfaz gráfica) que puede entonces imprimir los resultados.

El código de la clase Recorder está en el Anexo D.

5.2.2. Procesar la señal: el pitch tracker

Toda la lógica de detección de tono está dentro de una clase llamada PitchDetector, localizada en logic/sound. Contiene un método llamado getPitch(), que recepciona el buffer leído desde el mícrófono e invocado de manera asíncrona, para que esta lógica se ejecute al mismo tiempo que el refresco de la vista, que es bastante pesado. getPitch() consiste en una implementación del algoritmo Fast-Lifting-Wavelet-Transform basada en las wavelets de Haar, algoritmo presentado en la parte 2, apartado 2.2.3.

La implementación se implementa con operaciones sencillas (comparaciones, adiciones, diferencias, divisiones por 2) que usan solo tipos primitivos de Java: los int y double. Hay una diferencia importante entre los objetos Integer y los int, igual que entre los Double y los double. Por un lado están los wrappers, conteniendo métodos útiles tipo toString(), por otro lado son los tipos primitivos que solo contienen un valor, y no son instancias de ninguna clase, y las operaciones cuestan un poco menos.

El código completo de esta clase se puede encontrar en los Anexos, anexo C y está implementada en base a la implementación Matlab encontrada en el paper accesible desde la bibliografia: [1]

Sin embargo, detectar el tono no es suficiente y es recomendable compararlo con el tono previo. En efecto, un error común de los detectores de tono es una división o multiplicación por dos de la frecuencia real, debido al problema de fundamental aparente explicado en la sección 2 apartado 2.1.2. Por eso, se creó otra función cuya responsabilidad es la de realizar una comparación según la siguiente lógica: si el tono está muy cerca (menos del 20 % de diferencia en su valor) del previo, es un valor probable fiable. Si no, hay que ver si corresponde a la mitad, o al doble del valor previo, aproximadamente. Si es el caso, es probable que se deba a un error de octava y hay que dar un resultado consistente (generalmente seguir el valor previo, es lo que se hace en este trozo de código). Si el valor no encaja en esos resultados, entonces es un valor 'suelto' y se descarta debido a que la voz humana y el canto humano, como muchos fenómenos naturales, no puede contener discontinuidades abruptas y la voz no puede saltar 3 o 4 octavas en 50ms (resolución actual del detector de tono), incluso para los cantantes profesionales.

5.2.3. Mostrar/Dibujar los resultados

Dibujar una gráfica representativa de la afinación, en tiempo real, no es nada obvio y el diseño fue pensado de manera esquemática, adoptando una estética sin ninguna clase de chartjunk [1] y lleva colores evocadoras (rojo: lejos de la afinación correcta, verde, correcto).

Se decidió implementar un piano roll. Un piano roll es una gráfica que contiene un teclado de piano, y, con el tiempo, desfilan las notas como puntos hasta o desde su tecla correspondiente, horizontalmente o verticalmente.

Para tener un cierto nivel de control, y ser fiel a los principios de modularidad enunciados en la introducción, varias se han creado varias abstracciones para producir un código mantenible y fácil de leer.

Estas abstracciones están en la carpeta ui puesto que son abstracciones gestionando elementos gráficos, como puntos, gráficas, y teclado virtual. Son las siguientes:

- PianoChart: una clase que extiende la clase Android SurfaceView, que provee una área de píxeles sobre la que dibujar, conteniendo todos los objetos Point, y el Piano, gestionando el thread responsable de la actualización de la vista, el ViewThread
- ViewThread: una clase que extiende de la clase Java Thread, cuya responsabilidad es la de, con un número de actualizaciones (FPS) determinado, ordenar el refresco del PianoChart por el thread UI principal en cuanto está libre llamando a la función de dibujo del PianoChart, función conteniendo el rendering del Piano y de la gráfica (conjunto de Points).
- Point: una clase que contiene todos los datos necesarios para dibujar un punto. Sus coordenadas, y componentes de color R, G, B, alpha.
- Piano: una clase que contiene métodos para gestionar y renderizar las teclas Key

[1] El chartjunk se refiere a un uso de símbolos, o procesos estilisticos superfluos que no aportan nada y pueden molestar la buena interpretación de la gráfica y de los datos que contiene, esta terminología está sacado de los principios de Gestalt incluídos en el libro [6]

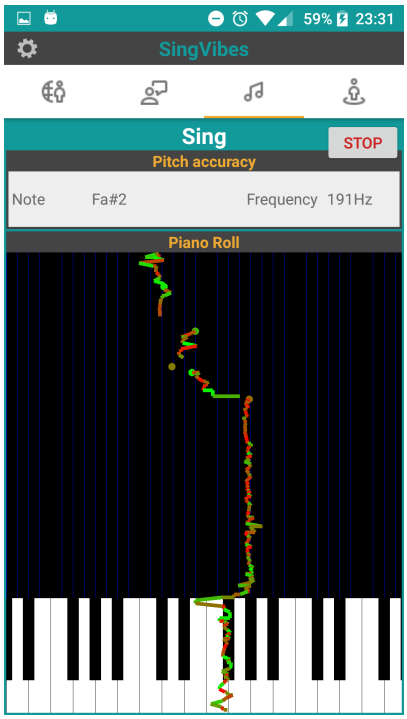


FIGURA 5.1: Vista durante la grabación y ejemplo de gráfica de la afinación en tiempo real

- Key: clase que encapsula toda la lógica acerca de una tecla, sus coordenadas, su color, y las instrucciones necesarias para su rendering

Referirse al anexo E para el diagrama de secuencia completo del proceso de grabación y refresco de las vistas.

La figura 5.1 muestra un ejemplo de resultado dibujado en tiempo real-

Cuando el Recorder invoca el método publishFrequency(), se almacena información sobre la nota y el tono detectado en un objeto llamado Note (conteniendo informaciones sobre la nota detectada: la frecuencia, amplitud media, tiempo cuando ocurrió, octava, semitono, número de tecla...). Este objeto Note se añade a un ArrayList<Note>(tabla ordenada de instancias de la clase Note en java). También se invoca el método addPoint() del PianoChart, que añade un punto en la gráfica a partir del objeto Note construido.

Todo este proceso se hace sobre el thread principal gráfico, dejando libre el thread (Runnable) de grabación y de procesado de las muestras de manera continua. Es interesante recordar que el refresco de los puntos (de sus coordenadas, color, etc) se hace de manera asíncrona en otro thread: el viewThread. Explicaciones se pueden encontrar más detalladas en el anexo E.

Estas abstracciones permitieron la implementación de funcionalidades complejas sin grandes dificultades como la posibilidad de mover el teclado (el eje 'x' que representa la frecuencia y el eje 'y' el tiempo) para poder ver la zona gráfica que interesa más mediante un listener sobre la superficie del PianoChart. Si el usuario toca la parte izquierda, la gráfica se traslada de manera continua con el teclado hasta la derecha para que se vea la parte de la gráfica oculta a la izquieeda. La misma lógica se aplica para ver la parte derecha oculta.

Esta implementación se hace mediante una variable que gestiona la velocidad y la translación sobre el eje X, actualizada continuamente por el ViewThread y por

la acción de usuario: tocar/no tocar la pantalla, mediante objetos Android OnTouchListener que proveen una manera de asociar la ejecución de un código después de un evento, aquí el evento OnTouch.

5.2.4. Convertir la señal en fichero: WAV y después comprimirlo

Una vez que la grabación se acaba, se desea guardar el fichero de sonido. Como se ha dicho antes, en Android no se puede a la vez procesar las muestras una por una y guardar el fichero puesto que no se usa la misma clase para hacer una u otra cosa (ver diferencia entre los MediaRecord y AudioRecord). Así que hay que reconstruir un fichero manualmente a partir de las muestras (que, recordamos, son de tipo integer, signed, de 16 bits (2 bytes), y contienen la amplitud de la señal).

La primera etapa es la construcción de un fichero no comprimido, un fichero .WAV. Sin entrar en los detalles, un fichero WAV tiene una cabecera, y todas las muestras escritas de una cierta manera, secuencialmente en formato binario.

Se encontró un código Java que crear un fichero WAV desde sus muestras. El problema de un WAV reside en su tamaño, que podemos calcular fácilmente del siguiente modo (para un WAV mono, con un solo canal):

$$size = headerLength + sampleSize.sampleRate.durationInSeconds$$

Un header son 42 bytes (referirse a la especificación del formato WAV cuya referencia está en la bibliografía), en calidad CD el sampleRate es de 44100Hz, y se cuantifica con 16 bits por muestra, por tanto, cada una de las muestras necesita 2 bytes. Una grabación de 5 minutos ocupa 42 + 88200 * 300 = 26 460 042 bytes, aproximadamente 25 MB para un WAV mono.

Entonces, una vez creado el fichero WAV, para implementar completamente los objetivos de la aplicación, hay que comprimir con Android, el fichero WAV en un formato usual tipo MP3.

El problema principal es que MP3 no es un formato libre. Hay que utilizar una librería ya creada que lo permita. El plan, que no se pudo llevar a cabo a la hora de redactar esta memoria, es de utilizar el NDK de Android que proporciona una manera de ejecutar ciertas librerías precompiladas hechas en C++, para así ser capaz de ejecutar una versión alterada de una librería como ffmpeg como si se dispusiera de un acceso a un terminal para lanzar un ejecutable compilado.

5.2.5. Guardar los resultados y publicación hasta el servidor

Los ficheros de audio grabados (WAV por ahora, MP3 en el futuro) se guardan en una carpeta en la memoria interna del teléfono, por ahora una carpeta oculta que está localizada en la carpeta /data/data/'nombre del paquete de la aplicación'/files. El nombre de este fichero es el resultado de una función de hash que convierte la concatenación del id del usuario en el servidor del usuario con su correo electrónico y la fecha más la hora de la grabación. Eso permite asegurar que cada fichero, una vez mandado al servidor, tenga un nombre único, pese a ser generado por la aplicación. En efecto, el usuario puede decidir no compartir su grabación hasta un cierto tiempo y se queda entonces solamente almacenada en el teléfono de este usuario.

Después de eso, se instancia un modelo que representa la grabación, un Recording, que contiene los datos de la grabación: título de la canción, artista, descripción, el id del usuario que lo hizo, y un campo data que es:

Cada nota son 2 enteros y un double separados por coma: tiempo, intensidad, frecuencia detectada. Por ahora, este campo datos es una string.

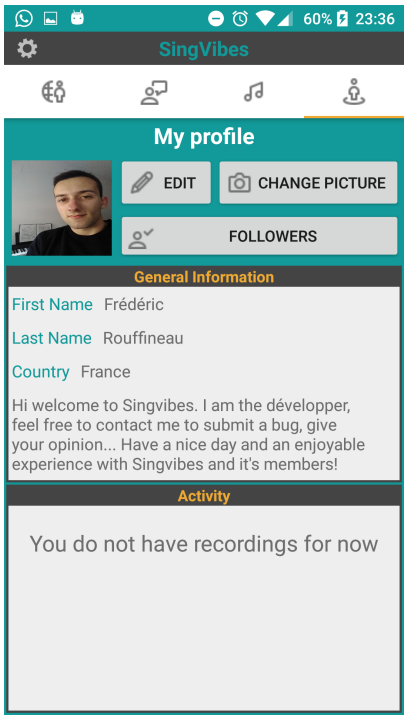


FIGURA 5.2: HomeFragment: vista del perfil de usuario

En un desarrollo posterior se espera que una nota sea un cadena binaria en el servidor, almacenada en un campo de tipo BLOB en MySQL, conteniendo un entero de 4 bytes, el tiempo en millisegundos, otro de 1 byte con la intensidad, y un double de 4 bytes para la frecuencia (9 bytes por nota).

Eso forma parte de las mejoras imprescindibles para poder hacer pública la aplicación que estarán presentadas en la ultima sección (6).

Un diagrama resumiendo todo el proceso desde el principio de la grabación hasta el post processing está en el Anexo E.

5.3. La implementación de la red social

5.3.1. Queries asíncronas, HTTPS y refresco de las vistas

En Android, como se ha dicho más arriba, el thread principal debe estar tan poco ocupado como sea posible para garantizar al usuario una experiencia fluida en la aplicación. Además, el móvil del usuario puede estar conectado o no, dependiendo del estado de la red y de la conectividad. Descargar los datos desde el servidor puede costar un tiempo variable, y entonces se debe ejecutar de manera asíncrona, y cuando los datos están disponibles es posible refrescar las vistas.

La figura 5.2 muestra la página de perfil personal. Las informaciones se cargan y se muestran poco a poco mientras que los datos se recuperan desde o el servidor, o la base de datos.

Este concepto está implementado con las siguientes clases:

- Las clases de la carpeta de la aplicación logic/network/queries: hay clases que gestionan las tareas asíncronas para recuperar los datos desde el servidor y una vez recuperados, se encargan de refrescar la base de datos y la vista. Extienden la clase AsyncTask del framework Android, conteniendo 2 métodos

muy importantes: doInBackground (ejecutado en un hilo por separado), y onPostExecute, ejecutado en el thread principal, usado para actualizar las vistas cuando se acaba la tarea ejecutada en el doInBackground. Para una lista de completa de las urls, referirse a la figura 3.1.

- Una tarea consiste en la llamada de un método llamado execute() de una instancia de la clase Task, definida en raiz/logic/network/Task.java. Este método utiliza los parámetros dados durante la instanciación, entre otros la url, los parámetros POST, que contienen siempre el token de sesión del usuario que permite al servidor identificar el usuario, y un campo llamado 'last_updated_at', utilizado para limitar los datos mandados por el servidor. El método execute() acepta un booleano que activa o no el almacenamiento de los datos recibido en el teléfono.

- El método Task::execute() invoca otra clase raiz/logic/network/HttpsRetriever.java, que tiene la responsabilidad de comprobar el estado de la conexión, ejecutar la petición HTTPS hasta el servidor, esperar la respuesta y devolver el JSON recibido.

- Si hay que refrescar la base de datos, se invoca el método refreshDB de la clase DBInterface (raiz/logic/db/DBInterface.java), dándole la url, y la respuesta JSON. Esta clase posee un método llamado jsonToDB, que crea peticiones SQL, con las mismas clases que LeafStormMVC adaptadas para funcionar con Java y SQLite, para almacenar los datos.

- Las actividades y componentes gráficos tienen entonces la posibilidad de instanciar modelos, (igual que LeafStormMVC), clase Model.java, para manipular colecciones de datos de un golpe.

5.3.2. Base de datos local

Para evitar un tráfico demasiado denso hasta el servidor, los resultados traídos desde el webservice se guardan en una base de datos SQLite y con una duración de expiración variable dependiendo del contenido. Cada vez que se necesitan datos, por ejemplo al hacer un clic para ver el perfil de un usuario, se buscan en la base de datos local. Si están, no se hace la petición hasta el servidor y se cargan los datos desde la base de datos. Si no, se actualizan los datos. Este mecanismo está implementado por una clase dedicada, la DBInterface, que contiene un método genérico llamado refreshDB, capaz de convertir el JSON devuelto por el servidor en ordenes SQL para que se almacene la información.

La lógica de los modelos del servidor presentadas en la sección 4 fue adaptada para funcionar también con Android. Se puede, a partir de una QueryFactory, recuperar un ArrayList de modelos (una colección ordenada de modelos que se pueden manipular de manera sencilla gracias a la implementación de java del patrón de diseño Iterator [2]).

[2]El patrón de diseño consiste en la implementación de un objeto conveniente que permite iterar sobre una colección de objetos de una clase determinada. Iterar significa recoger la colección uno por uno, con la posibilidad a cada etapa de acceder y ejecutar operaciones sobre este proyecto. Su forma más básica y visual está presente en cada implementación de los bucles de tipo foreach (para cada uno).

6 Resultados, conclusiones, trabajo futuro

6.1. Resultados

6.1.1. Una herramienta fácil de usar útil para visualizar el canto y encontrar a otros cantantes

A nivel funcional, la aplicación alcanza los objetivos propuestos. Permite cantar, visualizar con una gráfica en tiempo real la altura y la afinación del canto, grabar y escuchar las grabaciones, y poder utilizar un servicio de mensajería para hablar con otros miembros de la aplicación. Es una herramienta que se puede usar durante una sesión de entrenamiento de canto o vocalización y, desde la experiencia de uso personal, bastante cómoda. Los usuarios se pueden encontrar fácilmente mediante una funcionalidad de búsqueda, y aún más cuando las notificaciones estén disponibles, permitirá encontrar a gente que cantaron las mismas canciones que el usuario, por ejemplo.

6.1.2. Pitch tracking: desde el modelo hasta la implementación: enseñanzas y comparativa

El detector de tono fue unas de las partes más difíciles de implementar pese al hecho de tener disponible una base en código Matlab. Había que buscar bastante para encontrar un método que diera resultados consistentes y, sobre todo, que se pudiera calcular en tiempo real a una resolución adecuada. Para hacer esto en Java hubo que superar varios retos. Fue imprescindible Utilizar tipos primitivos y operaciones del nivel más bajo. Pero, no solo fue suficiente implementar el algoritmo. Toda una parte del trabajo residió en las pruebas con diferentes parámetros para obtener un resultado explotable. Por ejemplo, la idea intuitiva de utilizar la descomposición en series de Fourier es completamente impracticable en tiempo real. Y no es suficiente con aislar la fundamental (los armónicos a veces son más fuertes que la fundamental que, en algunos casos, ni siquiera está), pero calcular series de cosenos y senos es algo costoso, incluso utilizando las series de Taylor-MacLaurin. Utilizar las wavelets de Haar y el Fast-Lifting-Wavelet-Transform permite tener una resolución al menos 3 veces superior a utilizar la FFT (Fast-Fourier-Transform), y además se entiende mejor. Fue una buena enseñanza ver que la teoría matemática y conocimientos físicos son imprescindibles, pero es necesario disponer también de un cierto nivel de conocimientos a la hora de la implementación, no solo basta con 'hacer cálculos'.

6.1.3. LeafStormMVC + App Android: base para otras aplicaciones

Unos de los resultados más importantes de este proyecto es el hecho de tener un framework, un conjunto de clases, funcional y usable para cualquier otro proyecto. En efecto, LeafStormMVC está diseñado para poder soportar aplicaciones web

de todo tipo (preferencialmente de tamaño reducido, porque si no mejor usar un framework más grande que tenga más funcionalidades y se evita así 'reinventar la rueda', por ejemplo Magento para las tiendas en linea).

Programar para Android tampoco es algo trivial para obtener una aplicación estable. En efecto, el hecho de que todas las computaciones gráficas se deben ejecutar sobre el thread principal hace que la gestión de los hilos debe estar pensada en una manera completamente inversa a la mayoría de las aplicaciones, por ejemplo las escritas en C++ utilizando los frameworks Qt o GTK. La estructura de código que fue encontrada para separar de manera razonablemente limpia la lógica de la parte gráfica, y la ejecución asíncrona de las peticiones en Https del resto son cosas que serán reusadas en próximos proyectos de magnitud más grande.

Más de un 70 % de las clases desarrolladas pueden usadas tal cual en cualquier otro proyecto involucrando una aplicación Android conectándose a un webservice, y aseguran una implementación robusta del paradigma cliente-servidor, y muy extensible. Referirse a los repositorios ajuntados, anexo G.

6.2. Areas de mejora

6.2.1. Fallos del pitch tracker

El pitch tracker funciona bastante bien cuando hay una sola voz, y poco ruido ambiente. Si no, se comporta de manera completamente caótica, aunque se tomaron varias medidas para limitar los errores, como medir la intensidad de los sonidos entrantes y solo tomar el tono detectado cuando la media sobre todo el bloque de samples es superior a un valor que se puede determinar dentro de la aplicación.

Está adaptado también para una voz humana. Un sonido tan lleno de harmónicos como uno del piano puede poner en dificultad el detector de tono. Se podría mejorar mediante una análisis y una comparación de los tonos previos, y hacer una interpolación. Pero, no hay que olvidar el requisito del tiempo real, que genera muchas restricciones.

En el caso de un ruido ambiente continuo como el de un coche el detector es muy poco usable. Finalmente, un verdadero cantante usa mucho lo que se denomina el 'vibrato'. El tono oscila al alrededor de la nota que se quiere alcanzar para dar un efecto interesante sobre todo al final de algunas frases melódicas. Tal como está ahora, el detector de tono es incapaz de saber si las vibraciones son intencionales o si son el resultado de una mala técnica/afinación. Para obtener un resultado mucho más fiable, no se puede descartar la idea de utilizar una técnica de aprendizaje, de tipo deep learning, por refuerzo, alimentado manualmente por cantantes profesionales por ejemplo, comparando las grabaciones entre ellas dependiendo del nivel del usuario.

6.2.2. Seguridad

La seguridad de la aplicación por ahora cuenta mucho con el protocolo SSL y el certificado obtenido utilizando el método de Let's Encrypt, servicio gratis. Si el ssl está roto o un hackeador intercepta los datos y los decripta, se podría alterar la cuenta de usuario que interceptó (mediante un ataque de tipo Man In The Middle)

Una mejora posible sería implementar el protocolo OAuth 2 para asegurar las sesiones. Pero tampoco está exento de fallos.

Lo bueno de la aplicación es que no lleva datos sensible, lo que limita los daños de un posible ataque. Sin embargo, no se ha podido llegar a una solución contra los ataques por DDoS (Denial Of Service).

6.2.3. Normalización del código

Unos de los objetivos de este proyecto era también la realización completa de una aplicación web, con un enfoque pedagógico. Es uno de los elementos que motivaron la decisión de implementar muchas cosas a mano, hasta el router y el autoloader del servidor.

Al no ser profesional y completamente full stack aún, algunas normas no están cumplidas y unas de las próximas etapas de este proyecto será la normalización del código, por ejemplo aplicando las normas PSR para el código php que da especificaciones ayudando a la comprensión del código y limitando los fallos y las vulnerabilidades, permitiendo una cierta estandardización muy útil a la hora de colaborar con un equipo.

6.2.4. Mejor gestión de los intercambios de datos y caches

Unos de los objetivos, a largo plazo, de este proyecto, es hacer que cada dato del servidor sea mandando una sola vez a cada usuario, mediante el sistema de replicación progresivo de base de datos introducido en la sección 5, que permite limitar el número de peticiones hasta el servidor y entonces la carga.

La integración con RabbitMQ está planeada para el mes próximo que permite dejar abiertos canales de comunicación y evitar este refresco periódico y permitiendo un verdadero tiempo real y por tanto un ahorro considerable de carga servidor. RabbitMQ es un sistema de mensajería, consistiendo en un servidor RabbitMQ, y clientes para varios tipos de clientes, la mayoría de las tecnologías siendo soportadas (entre otros Java/Android y PHP). La desventaja es que RabbitMQ debe tener una máquina adecuada para funcionar bien porque mantendría abierta un canal por usuario autentificado y es difícil conseguir una estimación de las características necesarias del servidor para este proyecto sin tener una idea sobre el numero de usuarios que descargarían la aplicación una vez puesta en el Google Play.

Se considera también integrar Redis a LeafStormMVC, el sistema de cache por base de datos muy útil para los sitios internet que tienen un front end particularmente denso y el almacenamiento de las sesiones de usuario.

6.2.5. Funcionalidades adicionales y ventaja competitiva sobre las otras aplicaciones

Muchas funcionalidades han quedado por implementar para alcanzar todos los objetivos que surgieron durante el diseño y el desarrollo de la aplicación. Estas incluyen:

- El upload y la descarga en streaming de las grabaciones públicas de los usuarios (que pueden elegir qué grabación publicar o no)
- Un sistema de notificaciones
- Mejoras a nivel de UI y de UX: señalar los nuevos mensajes...
- Activar el newsfeed (depende del upload de las grabaciones)
- Proponer más opciones para controlar el algoritmo y la visualización
- Utilizar el NDK de Android para poder convertir el detector de tono en librería C++ precompilada e integrarla para ganar en eficiencia

- Poder convertir las grabaciones en fichero MIDI para que se puedan usar para escribir una partitura
- Poder tener otro tipo de visualización, como una partitura
- Recuperar las letras de las canciones e imprimirlas
- Poner un informe mucho más detallado acerca de una grabación, con análisis de ritmo por ejemplo
- Poner un sistema de puntuación que permite a un usuario de comparar sus grabaciones
- Poder superponer la gráfica de otra grabación a la que se está construyendo para poder tener una referencia que seguir

6.3. Plan de explotación de la aplicación

6.3.1. Google play, publicidad

Esta aplicación, una vez acabada, tiene la ventaja de tratar de un tema que interesa mucho, las aplicaciones de música y de canto suelen generar muchas descargas y tienen mucho éxito. Hay muchas maneras de diferenciarse de las que existen, y, se pueden utilizar varias aplicaciones sin exclusividad. Así qué, el primer paso para hacer una aplicación Android pública es subir la aplicación a la plataforma Google play. Dependiendo de la calidad de la aplicación, los usuarios pueden atribuir notas y comentarios ayudando al posicionamiento de la aplicación en los listados de Google.

La publicidad es una realidad común en las aplicaciones, y no se descarta incluir anuncios discretos en la aplicación automatizados tipo Google Ads que permiten generar una pequeña cantidad de dinero cada vez que un usuario ve un anuncio.

La integración con otros redes sociales como Facebook parece también una necesidad para alcanzar más gente y dejar la libertad de compartir sus resultados allí, y al mismo tiempo hacer que más gente conozcan a Singvibes.

6.3.2. Funcionalidades premium para financiar el servidor

El dinero no es la prioridad de este proyecto, antes de todo, este proyecto es pedagógico. Sin embargo, un servidor no es gratis. Un Amazon S3, quizás combinado con un Glacier para el contenido que apenas genera trafico, va a ser necesario para alojar las grabaciones de los usuarios y las imágenes. Combinado a una instancia EC2 decente, la factura puede rápidamente subir, hasta unos 100 euros al mes sin mucho problema para una aplicación que no es profesional y mucho más si tiene éxito. Así que, para limitar los riesgos, la idea sería implementar un sistema de pago por uso. Si el usuario quiere alojar mas que 30Mb de grabaciones más imágenes, tendrá que suscribir a un abono anual por ejemplo de 5 euros más o menos, lo necesario para amortiguar el coste de almacenamiento, junto a la publicidad. Esas decisiones no se pueden tomar sin saber el éxito que puede tener la aplicación. Sin embargo, se debe anticipar para evitar el escenario de, por ejemplo Pokémon GO, que funcionó mal durante todo el verano por haber subestimado el trafico.

6.3.3. Una escena abierta

Otra manera de disfrutar de un posible éxito de este proyecto es promocionar a los cantantes o a los trabajos musicales. Ha habido estos últimos años un cambio drástico en la mente de la gente y sobre todo de los jóvenes, que usan de manera

intensa las redes sociales como Facebook o Instagram para compartir absolutamente todo de lo que se les ocurre. Eso, también puede pasar por el canto. Quizás hacer de Singvibes el Instagram del canto, un poco como Smule intenta hacerlo, pero con una dimensión mucho más colaborativa y enfocado a la mejora de los principiantes. Eso, para una disciplina como el canto que exige un alto nivel de confianza y de apoyo parece importante y puede haber una oportunidad de lanzar una nueva clase de actividad. Un coaching vocal colaborativo donde todos ganan. Los que ayudan ganan en respeto, estima, y visibilidad para sus trabajos, y los principiantes ganan bastante confianza para empezar en serio el canto, con la ayuda de un profesor por ejemplo. Está aplicación no pretende sustituir clases profesionales de canto, pero quizás podría dar la chispa que falta a algunos para probar el canto o interesarse en la música en general.

Bibliografía

[1] Eric Larson, Ross Maddox *Real-Time Time-Domain Pitch Tracking Using Wavelets* https://courses.physics.illinois.edu/phys406/NSF_REU_Reports/2005_reu/Real-Time_Time-Domain_Pitch_Tracking_Using_Wavelets.pdf Illinois Physics Research Department, 2015

[2] Daubechies, Ingrid and Wim Sweldens *Factoring Wavelet Transforms into Lifting Steps* Anal. Appl, 1998

[3] Bernard Salamito, Stéphane Cardini, Damien Jurine, Marie-Noëlle Sanz *Physique Tout-en-un MPSI-PTSI*. (Francés) [*Manual de Física para alumnos de MPSI-PTSI*]. Dunod, 2016.

[4] Kevin Beaver *Hacking for Dummies, 5th edition*. For Dummies, December 21, 2015.

[5] A. Dannhauser *Teoría de la música*. Ricordi Americana, Sociedad Anónima Editorial y Comercial

[6] Wolfgang Kohler,*Gestalt Psychology: The Definitive Statement of the Gestalt Theory*. Liveright, January 17, 1970

[7] *Manual oficial PHP*, http://www.php.net/manual/en/

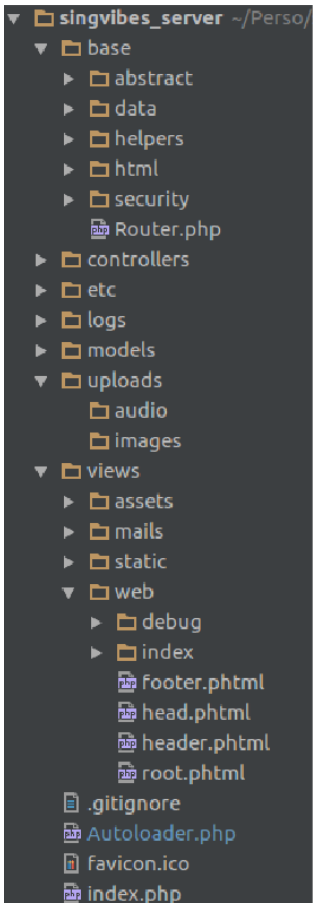
[8] *PHP Standards Recommendations: PSR*, http://www.php-fig.org/psr/

[9] *Documentación oficial Android*, https://developer.android.com/reference/packages.html

[10] *Stackoverflow*: más de 100 problemas diferentes, http://stackoverflow.com/

[11] Mitchell Anicas, *How To Secure Nginx with Let's Encrypt on Ubuntu 14.04* https://www.digitalocean.com/community/tutorials/how-to-secure-nginx-with-let-s-encrypt-on-ubuntu-14-04, para Digital Ocean, December 17, 2015

A Árbol de las carpetas/clases del servidor y de LeafStormMVC



La parte central de LeafStormMVC: no depende del proyecto, una vez acabada no se debería modificar

Contiene las clases abstractas de modelos y controladores para implementar el **Modelo-Vista-Controlador**

Interfaz con la base de datos y gestión de las operaciones CRUD, contiene un constructor de peticiones SQL

Clases genéricas que pueden ser invocadas desde cualquier sitio, como el Logger o el Mailer

Clases ayudando a la creación de componentes html como los form en los ficheros .phtml

Contienen las clases gestionando nociones claves en seguridad como las sesiones y los tokens de sesión

El **Router,** elige y ejecuta la acción de controlador apropiada dependiendo de la URL y de sus parámetros

Los controladores del paradigma MVC, organizados por frontname (web o api)

Contiene varis ficheros de configuración, los ficheros cacheados y los scripts de creación de la base de datos

Los modelos del paradigma MVC, extiendo de base/abstract/ModelAbstract.php

Las vistas del paradigma MVC,

Las imágenes, los css, javascript

Las plantillas html de correos electrónicos

Páginas html estáticas sin ninguna clase de computo (404...)

Ficheros .phtml (html conteniendo inclusiones de código php), del frontname web, ordenados por controlador

.phtml usados por el controlador web/DebugController

.phtml usados por el controlador web/IndexController

Footer genérico

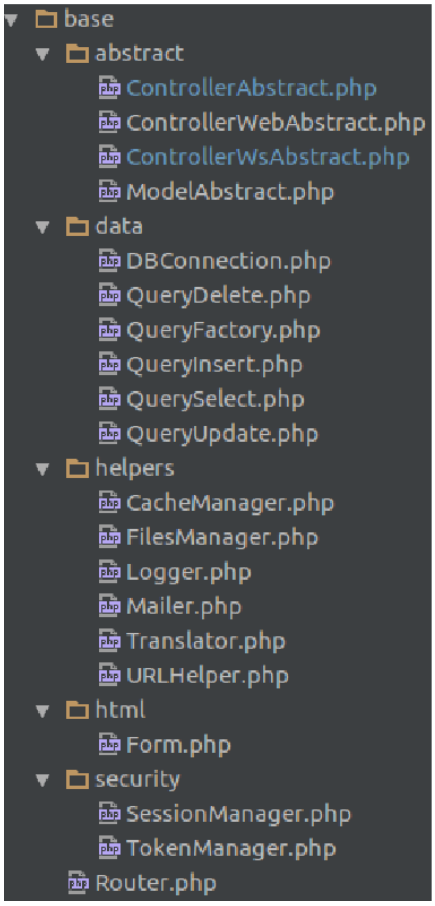
Parte <head> de la respuesta html, genérica

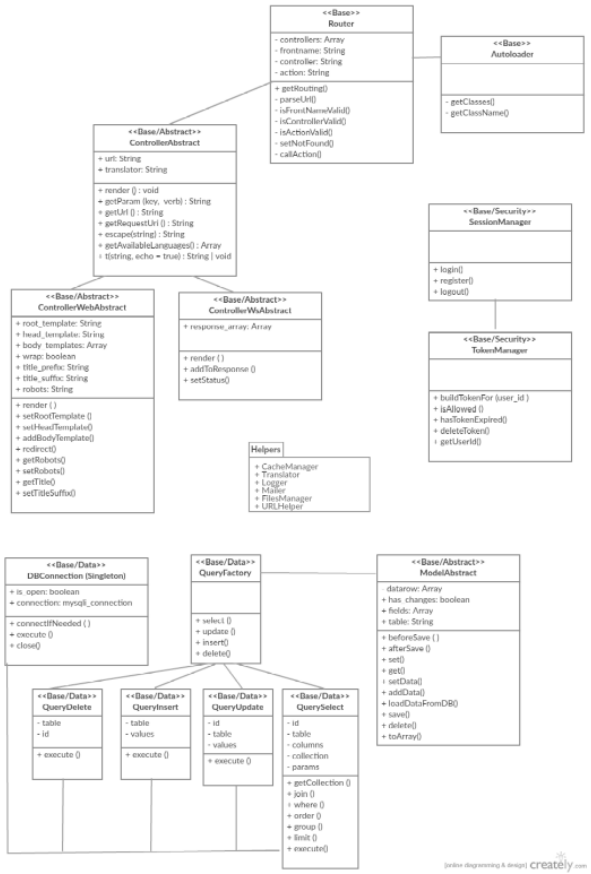
Header genérico

Estructura de la respuesta y de los templates .phtml invocados (head, header, cuerpo, footer)

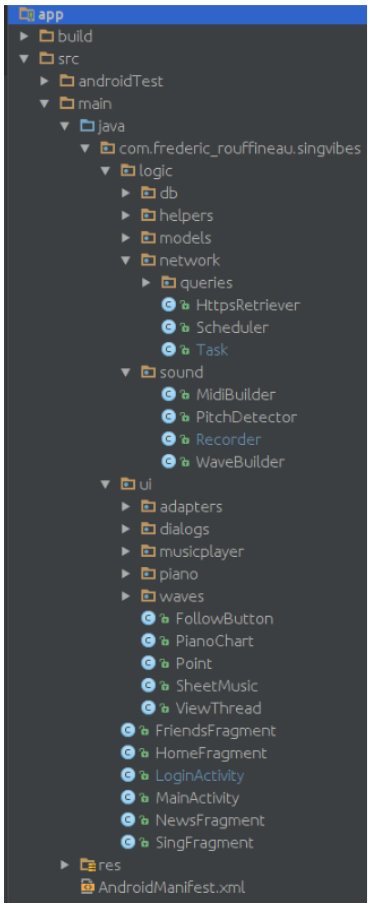
El Autoloader. Encargado de encontrar e incluír las clases cuando están invocadas, para solo cargar lo necesario

El Autoloader. Encargado de encontrar e incluír las clases cuando están invocadas, para solo cargar lo necesario


base
abstract
ControllerAbstract.php
ControllerWebAbstract.php
ControllerWsAbstract.php
ModelAbstract.php
data
DBConnection.php
QueryDelete.php
QueryFactory.php
QueryInsert.php
QuerySelect.php
QueryUpdate.php
helpers
CacheManager.php
FilesManager.php
Logger.php
Mailer.php
Translator.php
URLHelper.php
html
Form.php
security
SessionManager.php
TokenManager.php
Router.php


<<Base>>
Router
- controllers: Array
- frontname: String
- controller: String
- action: String
+ getRouting()
- parseUrl()
- isFrontNameValid()
- isControllerValid()
- isActionValid()
- setNotFound()
- callAction()
<<Base>>
Autoloader
- getClasses()
- getClassName()
<<Base/Abstract>>
ControllerAbstract
+ url: String
+ translator: String
+ render () : void
+ getParam (key, verb) : String
+ getUrl () : String
+ getRequestUri () : String
+ escape(string) : String
+ getAvailableLanguages() : Array
+ t(string, echo = true) : String | void
<<Base/Abstract>>
ControllerWebAbstract
+ root_template: String
+ head_template: String
+ body_templates: Array
+ wrap: boolean
+ title_prefix: String
+ title_suffix: String
+ robots: String
+ render ( )
+ setRootTemplate ()
+ setHeadTemplate()
+ addBodyTemplate()
+ redirect()
+ getRobots()
+ setRobots()
+ getTitle()
+ setTitleSuffix()
<<Base/Abstract>>
ControllerWsAbstract
+ response_array: Array
+ render ( )
+ addToResponse ()
+ setStatus()
<<Base/Security>>
SessionManager
+ login()
+ register()
+ logout()
<<Base/Security>>
TokenManager
+ buildTokenFor (user_id)
+ isAllowed ()
+ hasTokenExpired()
+ deleteToken()
+ getUserId()
Helpers
+ CacheManager
+ Translator
+ Logger
+ Mailer
+ FilesManager
+ URLHelper
<<Base/Data>>
DBConnection (Singleton)
+ is_open: boolean
+ connection: mysqli_connection
+ connectIfNeeded ( )
+ execute ()
+ close()
<<Base/Data>>
QueryFactory
+ select ()
+ update ()
+ insert()
+ delete()
<<Base/Abstract>>
ModelAbstract
- datarow: Array
+ has_changes: boolean
+ fields: Array
+ table: String
+ beforeSave ( )
+ afterSave ()
+ set()
+ get()
+ setData()
+ addData()
+ loadDataFromDB()
+ save()
+ delete()
+ toArray()
<<Base/Data>>
QueryDelete
- table
- id
+ execute ()
<<Base/Data>>
QueryInsert
- table
- values
+ execute ()
<<Base/Data>>
QueryUpdate
- id
- table
- values
+ execute ()
<<Base/Data>>
QuerySelect
- id
- table
- columns
- collection
- params
+ getCollection ()
+ join ()
+ where ()
+ order ()
+ group ()
+ limit ()
+ execute()

B Árbol comentado de las clases de la aplicación Android



Parte lógica de la aplicación que no interactua directamente con la UI
Interfaz con la base de datos y gestión de las operaciones CRUD
Grupos de funciones útiles que pueden ser llamadas por cualquier clase
Objetos y entidades manipuladas por la aplicación, tal como una nota, un usuario, una conversación...

Clases que gestionan el tráfico internet

Tareas **asincronas**: al menos una por url de API – ordena el refresco de la vista adecuada cuando se acaba
Construye y manda la petición **HTTPS**, se encarga de recibir el JSON y devolverlo a la Task que lo invocó
Una clase capaz de **ejecutar una secuencia de Task(s)** y planificarlas
Invoca el HttpsRetriever y la DBInterface para **refrescar la DB y da acceso al JSON devuelto**

Clases gestionando los **datos audio**: el grabador, el detector de tono y los constructores de ficheros
Constructor de ficheros MIDI (no implementado aún)
Detector de tono

Gestiona el micrófono y todo el proceso de grabación, ordena periodicamente el refresco de la vista
Construye ficheros WAV a partir de un conjunto de samples

Componentes gráficos o encapsulando lógica gráfica
Constructores de listas (listas Android), linea por linea
Pestañas popup conteniendo funcionalidad adicional (mensajeria, opciones...)

Piano: gestor del teclado y teclas
Clases no implementadas aún que permitirán visualisar datos acerca del sonido

Encapsula la lógica acerca de la gráfica de la afinación con el teclado de piano, extiende SurfaceView
Punto de la gráfica (contiene las coordenadas y datos sobre el color y la manera de dibujarlo)
Será otro tipo de representación en tiempo real: una partitura que se construye poco a poco
Hilo ordenando un refresco périodico de la gráfica durante el proceso de grabación

> **Clases fundamentales de Android gestionando el ciclo de vida de la aplicación,** que invocan todas las clases anteriores por capas, son asociadas a layouts, ficheros declarativos de Android que permiten declarar y configurar los componentes gráficos que se van a mostrar. Dichas funciones deben ser ejecutadas en el hilo principal de Android: el hilo gráfico

C Código del detector de tono: PitchDetector.java

```
package com.frederic_rouffineau.singvibes.logic.sound;
import com.frederic_rouffineau.singvibes.logic.helpers.Hmath;

/**
 * Created by frederic on 5/11/16.
 * Based on the paper:
    https://courses.physics.illinois.edu/phys406/NSF_REU_Reports/2005_reu/Real-Time_T
*/
public class PitchDetector {
   private double pitch;
   private double previousPitch = -1;
   private double pitchConfidence = -1;

   // WAVELET ALGORITHM CONFIGURATION
   public final static int maxFLWTlevels = 6;
   public final static double maxF = 3000.;
   public final static int differenceLevelsN = 3;
   public final static double maximaThresholdRatio = 0.75;
   private int samplingRate = 44100;

   // WAVELET STATE VARIABLES
   private double amplitudeThreshold; // Max amplitude authorized
       (max amplitude * k)
   private double samplesAvg;
   private int currentSamplesNumber;
   private double[] currentSamples;
   private int currentIteration;
   private int[] distances;
   private int[] mins;
   private int[] maxs;
   private double curModeDistance = -1.;

   public PitchDetector(int samplingRate) {
      this.samplingRate = samplingRate;
      this.resetPitchDetector();
   }

   public void resetPitchDetector() {
      this.previousPitch = -1;
      this.pitchConfidence = -1;
   }
   public double getPitch(double[] samples, double first, int
       length) {
      initializeState(length);
      prepareSamples(samples, first, length);
      computeAmplitudeThreshold();
```

```
while(this.currentIteration < maxFLWTlevels &&
   this.currentSamplesNumber >= 2) {
   if(this.currentIteration > 0) {
      resample(); // Extracts the approximation component
         from the samples
   }
   int delta = (int)
       (this.samplingRate/(Hmath.pow2(this.currentIteration)*maxF));
   double dv;
   double previousDV = -1000;

   int nbMins = 0;
   int nbMaxs = 0;
   int lastMinIndex = -1000000;
   int lastmaxIndex = -1000000;
   boolean findMax = false;
   boolean findMin = false;
   double si, si1;

   // LOOP: the idea is to find the mins and the maxs
   for (int i = 2; i < this.currentSamplesNumber; i++) {
      si = this.currentSamples[i] - samplesAvg;
      si1 = this.currentSamples[i-1] - samplesAvg;

      if (si1 <= 0 && si > 0) findMax = true;
      if (si1 >= 0 && si < 0) findMin = true;

      // min or max ?
      dv = si - si1;

      if (previousDV > -1000) {
         if (findMin && previousDV < 0 && dv >= 0) {
            // minimum
            if (Hmath.abs(si) >= this.amplitudeThreshold) {
               if (i > lastMinIndex + delta) {
                  mins[nbMins++] = i;
                  lastMinIndex = i;
                  findMin = false;
               }
            }
         }

         if (findMax && previousDV > 0 && dv <= 0) {
            // maximum
            if (Hmath.abs(si) >= this.amplitudeThreshold) {
               if (i > lastmaxIndex + delta) {
                  maxs[nbMaxs++] = i;
                  lastmaxIndex = i;
                  findMax = false;
               }
            }
         }
      }

      previousDV = dv;
   }
```

```
// If peaks are found
if (nbMins != 0 || nbMaxs != 0) {
   int d;
   distances = new int[length];
   for (int i = 0; i < nbMins; i++) {
      for (int j = 1; j < differenceLevelsN; j++) {
         if (i + j < nbMins) {
            d = Hmath.abs(mins[i] - mins[i + j]);
            distances[d] = distances[d] + 1;
         }
      }
   }
   for (int i = 0; i < nbMaxs; i++) {
      for (int j = 1; j < differenceLevelsN; j++) {
         if (i + j < nbMaxs) {
            d = Hmath.abs(maxs[i] - maxs[i + j]);
            distances[d] = distances[d] + 1;
         }
      }
   }

   // find best summed distance
   int bestDistance = -1;
   int bestValue = -1;
   for (int i = 0; i < this.currentSamplesNumber; i++) {
      int summed = 0;
      for (int j = -delta; j <= delta; j++) {
         if (i + j >= 0 && i + j <
             this.currentSamplesNumber)
            summed += distances[i + j];
      }
      if (summed == bestValue) {
         if (i == 2 * bestDistance)
            bestDistance = i;

      } else if (summed > bestValue) {
         bestValue = summed;
         bestDistance = i;
      }
   }

   double distAvg = 0.0;
   double nbDists = 0;
   for (int j = -delta; j <= delta; j++) {
      if (bestDistance + j >= 0 && bestDistance + j <
          length) {
         int nbDist = distances[bestDistance + j];
         if (nbDist > 0) {
            nbDists += nbDist;
            distAvg += (bestDistance + j) * nbDist;
         }
      }
   }
   distAvg /= nbDists;

   if (curModeDistance > -1.) {
```

```
                if (Hmath.abs(distAvg * 2 - curModeDistance) <= 2 *
                    delta) {
                    this.pitch = this.samplingRate /
                        (Hmath.pow2(this.currentIteration - 1) *
                        curModeDistance);
                    break;
                }
            }
            curModeDistance = distAvg;
        } else {
            break;
        }
        this.currentIteration++;
    }

    return this.adjustWithPreviousPitch(this.pitch);
}

public void initializeState(int length) {
    this.currentIteration = 0;
    this.pitch = 0.0; // Pitch final
    distances = new int[length];
    mins = new int[length];
    maxs = new int[length];
    curModeDistance = -1.;
}

public void prepareSamples(double[] samples, double first, int
    length) {
    length = Hmath.getFirstSuperiorPowerOf2(length);
    this.currentSamples = new double[length];
    for(int i=0; i<length; i++) {
        this.currentSamples[i] = samples[i] + first;
    }
    this.currentSamplesNumber = length;
}

private void computeAmplitudeThreshold() {
    this.samplesAvg = 0.0;
    double maxValue = 0.0;
    double minValue = 0.0;

    double sample;
    for (int i = 0; i < this.currentSamplesNumber;i++) {
        sample = this.currentSamples[i];
        this.samplesAvg = this.samplesAvg + sample;
        if (sample > maxValue) maxValue = sample;
        if (sample < minValue) minValue = sample;
    }

    this.samplesAvg = this.samplesAvg/this.currentSamplesNumber;
    maxValue = maxValue - samplesAvg;
    minValue = minValue - samplesAvg;
    this.amplitudeThreshold = (maxValue > -minValue ? maxValue :
        -minValue) * maximaThresholdRatio;
}
```

```
// This method takes only half of the samples, by converting
    each consecutive pair to its mean value
// This actually performs the FLTW by isolating the
    approximation from the detail in the signal
public void resample() {
   for (int i = 0; i < this.currentSamplesNumber/2; i++) {
      this.currentSamples[i] = (this.currentSamples[2*i] +
         this.currentSamples[2*i + 1])/2.;
   }
   this.currentSamplesNumber /= 2;
}

private double adjustWithPreviousPitch(double pitch) {
   // equivalence
   if (pitch == 0.0) pitch = -1.0;

   double estimatedPitch = -1;
   double acceptedError = 0.2f;
   int maxConfidence = 5;

   // If a pitch has been detected
   if (pitch != -1) {
      // If there is no previous pitch
      if (this.previousPitch == -1) {
         estimatedPitch = pitch;
         this.previousPitch = pitch;
         this.pitchConfidence = 1;
      // Else If the two pitches are less distant than 20%,
         raise confidence
      } else if (Hmath.abs(this.previousPitch - pitch)/pitch <
         acceptedError) {
         this.previousPitch = pitch;
         estimatedPitch = pitch;
         this.pitchConfidence = Hmath.min(maxConfidence,
            this.pitchConfidence + 1); // maximum 3
      // Else, if the confidence level is high enough, and the
         same applyes to this pitch multiplied per 2 (IE:
         octava error)
      } else if ((this.pitchConfidence >= maxConfidence-2) &&
         Hmath.abs(this.previousPitch - 2.*pitch)/(2.*pitch) <
         acceptedError) {
         // close to half the last pitch, which is trusted
         estimatedPitch = 2.*pitch;
         this.previousPitch = estimatedPitch;
      // Else, if same goes for the pitch divided by two
      } else if ((this.pitchConfidence >= maxConfidence-2) &&
         Hmath.abs(this.previousPitch - 0.5*pitch)/(0.5*pitch)
         < acceptedError) {
         // close to twice the last pitch, which is trusted
         estimatedPitch = 0.5*pitch;
         this.previousPitch = estimatedPitch;
      // Else, the value has nothing to do with it
      } else {
         // nothing like this : very different value
         if (this.pitchConfidence >= 1) {
            // previous trusted : keep previous
            estimatedPitch = this.previousPitch;
```

```
                this.pitchConfidence = Hmath.max(0,
                    this.pitchConfidence - 1);
            } else {
                // previous not trusted : take current
                estimatedPitch = pitch;
                this.previousPitch = pitch;
                this.pitchConfidence = 1;
            }
        }
    // Else, if no pitch has been detected this time
    } else {
        // If there was a pitch before and it is trusted enough,
            take it
        if (this.previousPitch != -1) {
            if (this.pitchConfidence >= 1) {
                estimatedPitch = this.previousPitch;
                this.pitchConfidence = Hmath.max(0,
                    this.pitchConfidence - 1);
            } else {
                this.previousPitch = -1;
                estimatedPitch = -1.;
                this.pitchConfidence = 0;
            }
        }
    }

    if (this.pitchConfidence >= 1) {
        pitch = estimatedPitch;
    } else {
        pitch = -1;
    }

    if (pitch == -1) { pitch = 0.0; }
    return pitch;
  }
}
```

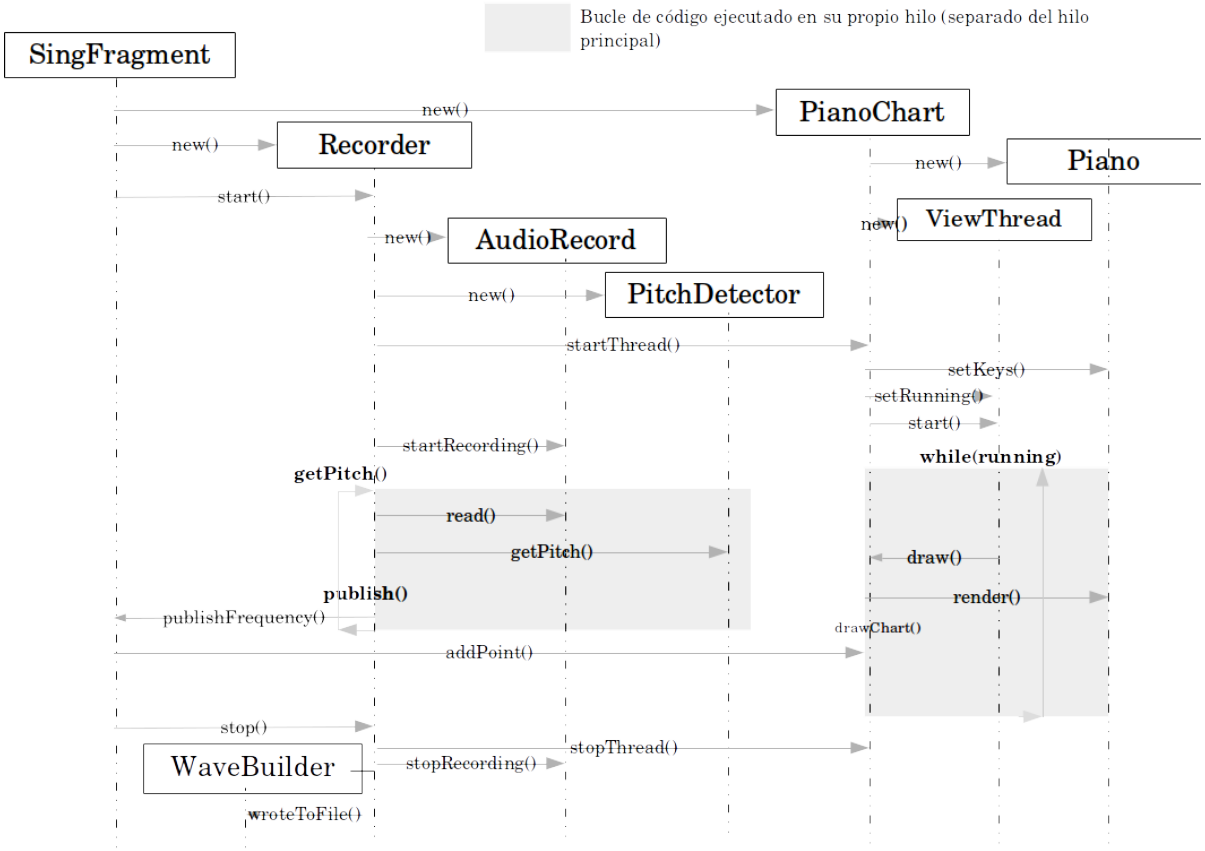
D Código del grabador: Recorder.java

```
package com.frederic_rouffineau.singvibes.logic.sound;

/**
 * Created by frouffineau on 8/11/16.
 *
    https://www.newventuresoftware.com/blog/record-play-and-visualize-raw-audio-data-
 *
    http://stackoverflow.com/questions/8499042/android-audiorecord-example
 *
    http://stackoverflow.com/questions/11985518/android-record-sound-in-mp3-format
 */

import android.media.AudioFormat;
import android.media.AudioRecord;
import android.media.MediaRecorder;
import android.os.Handler;
import android.util.Log;

import com.frederic_rouffineau.singvibes.MainActivity;
import com.frederic_rouffineau.singvibes.SingFragment;
import com.frederic_rouffineau.singvibes.logic.helpers.Hfiles;
import com.frederic_rouffineau.singvibes.logic.helpers.Hmath;
import com.frederic_rouffineau.singvibes.ui.PianoChart;

import java.io.FileOutputStream;
import java.util.ArrayList;


public class Recorder {
   public final static int sampleRate = 44100;
   public final static int audioSource =
      MediaRecorder.AudioSource.MIC; // Audio source is the device
      MIC
   public final static int channelConfig =
      AudioFormat.CHANNEL_IN_MONO; // Recording in mono
   public final static int audioEncoding =
      AudioFormat.ENCODING_PCM_16BIT; // Records in 16bit
   public final static int blockSize = 1024; // must be a power of
      two
   public final static int timeInterval = 50;
   public final static double MINIMUM_INTENSITY = 1000;

   private MainActivity activity;
   private SingFragment singFragment;

   private boolean isRecording = false;
```

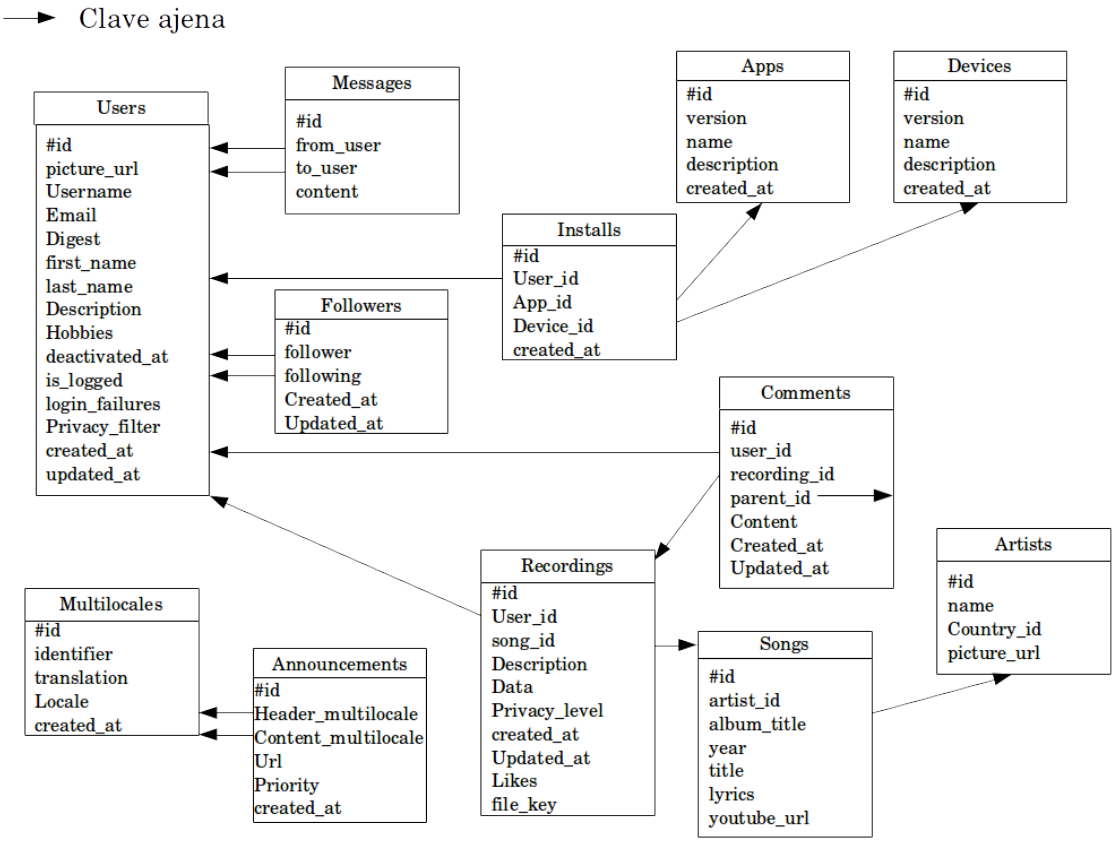
```
private long startRecording = 0;
private AudioRecord audioRecord;
private PitchDetector pitchDetector;
private double f;
private double intensity;
private Handler handler = new Handler();
private WaveBuilder waveBuilder;
private ArrayList<Short> pcm_array;
private PianoChart pianoChart;

public Recorder(MainActivity a, SingFragment sf, PianoChart pc)
    {
   this.activity = a;
   this.singFragment = sf;
   this.pianoChart = pc;
}

public boolean isRecording() {
   return isRecording;
}

public void start() {
   Log.i("INFO", "Recording start");
   if(!this.isRecording){
      this.isRecording = true;
      try{
         int bufferSize =
            AudioRecord.getMinBufferSize(sampleRate,
            channelConfig, audioEncoding);
         this.audioRecord = new AudioRecord(audioSource,
            sampleRate, channelConfig, audioEncoding,
            bufferSize);
         this.pitchDetector = new PitchDetector(sampleRate);
         pianoChart.startThread();
         pcm_array = new ArrayList<>();
         Log.i("INFO", "Recording start");
         audioRecord.startRecording();
         this.startRecording = System.currentTimeMillis();
         handler.postDelayed(getPitch, timeInterval);
      }
      catch(Exception e){
        e.printStackTrace();
      }
   }
}

public void stop() {
   if(this.isRecording && this.audioRecord != null){
      this.isRecording = false;
      pianoChart.stopThread();
      audioRecord.stop();
      Short[] pcma = pcm_array.toArray(new
         Short[pcm_array.size()]);
      waveBuilder = new WaveBuilder(sampleRate, (short) 1,
         pcma, 0, pcm_array.size());
      waveBuilder.wroteToFile(Hfiles.getFilePath(this.activity,
         "tmp.wav"));
```

```
            //waveBuilder.convertToMP3("tmp.wav", this.activity);
        }
    }

    Runnable getPitch = new Runnable() {
        public void run() {
            short[] buffer = new short[blockSize];// Save the raw PCM
                samples as short bytes
            double[] samples = new double[blockSize];
            int bufferReadResult =
                Recorder.this.audioRecord.read(buffer, 0, blockSize);
            double samplesLength = Hmath.min(blockSize,
                bufferReadResult);
            for (int i = 0; i < samplesLength; i++) {
                samples[i] = (double) buffer[i]; // signed 16 bit
                intensity += Hmath.abs(samples[i]);
                pcm_array.add(buffer[i]);
            }
            intensity /= samplesLength;
            f = pitchDetector.getPitch(samples, samples[0],
                blockSize);

            activity.runOnUiThread(publish);
            if(isRecording) {
                handler.postDelayed(getPitch, 0);
            }
        }
    };

    Runnable publish = new Runnable() {
        public void run() {
            Long s = System.currentTimeMillis() - startRecording;
            singFragment.publishFrequency(f, s, intensity);
        }
    };
}
```

E Diagrama de secuencia general del proceso de grabación



F Esquema de la base de datos con las claves ajenas



G Repositorios de código y aplicación

El código completo del proyecto está disponible en Github y seguirá evolucionando.

- https://github.com/fredrfn/Singvibes-Server
- https://github.com/fredrfn/Singvibes-For-Android

Además, se puede encontrar más información en el sitio oficial de la aplicación: https://singvibes.com. Para cualquier información, o si se desea señalar un bug, se puede usar el formulario de contacto https://singvibes.com/contact, o abrir una issue en Github.

Índice de figuras