



Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica



Isolaatti

Aplicación de Android

Documentación

Ingeniería de Dispositivos Móviles

Ing. Héctor Hugo Flores Moreno

Jueves N1-N3

Estudiante: Erik Everardo Cavazos Hernández

Matrícula: 1811290

Semestre: Enero – junio 2020

A. IDENTIFICACIÓN DEL PROYECTO

Fecha de Elaboración: 13/02/2020

1. Nombre del Proyecto:	Isolaatti: aísla los instrumentos de una canción
2. Unidad(es) de Aprendizaje:	Ingeniería de Dispositivos Móviles
3. Profesor(es) responsable:	Ing. Héctor Hugo Flores Moreno
4. Nombre del Empresa:	
5. Estudiante:	Erik Everardo Cavazos Hernández
6. Profesores vinculados al proyecto:	

B. DESCRIPCIÓN DEL PROYECTO

1. Descripción general del proyecto:	Una aplicación que separa los instrumentos de una canción, de tal manera que se pueda escuchar de manera aislada: batería, bajo, voces y otros (pudiendo ser guitarras, pianos, sintetizadores, metales, etc.)
2. Objetivo(s) del proyecto	<ul style="list-style-type: none"> • Que la aplicación sea usable por cualquier persona sin dificultad. • Cumplir con la función principal
3. Propósito del proyecto	<ul style="list-style-type: none"> • Llegar a usuarios de Android y poder obtener ingresos (con anuncios), para poder mantener el hosting de los servidores y no perder dinero al menos.
4. Herramientas de desarrollo y/o Materiales	<p>IDEs y editores de código:</p> <ul style="list-style-type: none"> • Android Studio • Visual Studio Code • JetBrains Rider <p>Lenguajes de programación:</p> <ul style="list-style-type: none"> • Kotlin (app de Android) • Python (script del servidor de proceso) • C# (WEB API)

	<p>Base de datos</p> <ul style="list-style-type: none"> • MySQL • SQLite (base de datos local en app) <p>Software externo</p> <ul style="list-style-type: none"> • https://github.com/facebookresearch/demucs <p>Librerías, frameworks y SDKs</p> <ul style="list-style-type: none"> • Flask (Python) • .NET Core 3.0 • Android SDK <p>Control de versiones</p> <ul style="list-style-type: none"> • Git (GitHub, remoto) <p>Otros</p> <ul style="list-style-type: none"> • Terminal • SSH
5. Estimación del costo del proyecto	<p>El proceso de desarrollo no costará dinero, sin embargo, implementarlo sí.</p> <p>El costo varía dependiendo del numero de servidores de proceso y la capacidad.</p> <p>Ejemplo:</p> <p>Una maquina virtual con 2 nucleos y 8 GB de RAM: 85 USD por mes.</p> <p>Para fines de pruebas se usarán varios <i>Azure App Services</i>, los cuales son gratis, pero con menos capacidad de proceso.</p> <p>Durante el desarrollo basta con usar una PC</p>
6. Referencias de proyectos anteriores	<p>El proyecto no sería posible sin:</p> <p>https://github.com/facebookresearch/demucs</p> <p>El cual funciona solo a través de línea de comandos. Este proyecto es implementado en un servidor WEB.</p>

7. Innovación en el proyecto	Crear un cliente Android con una interfaz gráfica fácil de usar, el cual permita fácilmente obtener las pistas de una canción.
------------------------------	--

C. PLANEACIÓN DE LAS ACTIVIDADES

Actividades	Fechas
Escribir el script en Python que se encargará de recibir los archivos de audio, y procesarlos para obtener las cuatro pistas.	13 de febrero
Desarrollar la API	
Base de datos	20 de febrero
Registro de cuentas	20 de febrero
Inicio de sesión	20 de febrero
Controlador para recibir peticiones del servidor de proceso, las cuales indicarán cuando un archivo ya ha sido procesado, con lo cual se procederá a guardar la URL de cada pista en la base de datos, así como enviar una notificación al cliente (por el sistema de notificaciones de Android, y por correo electrónico)	20 de febrero
Método para enviar correo electrónico al usuario cuando su archivo esté listo	27 de febrero
Despliegue de API	27 de febrero

Implementación en la nube del script mencionado en la primera actividad	27 de febrero
Desarrollo de la aplicación Android	
Empezar el proyecto de Android Studio y configurar parámetros	27 de febrero
Pantalla de registro de cuentas y inicio de sesión	5 de marzo
Pantalla de carga de archivos	5 de marzo
Base de datos SQLite para guardar preferencias y otras cosas en local	12 de marzo
Pantalla de consulta de archivos (canciones) procesados	12 y 19 de marzo
Proceso en segundo plano (el que lance la notificación cuando un archivo en espera ha sido procesado y está listo para ser descargado)	26 de marzo y 2 de abril
Reproductor para poder reproducir las canciones y silenciar o no las pistas que se deseen	2 y 23 de abril
Método de envío de notificaciones	7 de mayo
Pruebas con usuarios	14 de mayo
Correcciones de errores	14 y 21 de mayo

Puesta a punto para Play Store	26 de mayo
--------------------------------	------------

D. DESARROLLO DEL PROYECTO

1. Investigación y/o Análisis del Proyecto	Mi aplicación hará uso del modelo cliente-servidor. Se basa en una o más listas de espera.
2. Diseño del Proyecto	<p>El proyecto consiste en desarrollar un sistema, compuesto por tres partes mayores:</p> <ul style="list-style-type: none"> • El cliente Android • Una WEB API • N servidores de proceso <p>Resumen</p> <ol style="list-style-type: none"> 1. Se registra en inicia sesión (solo la primera vez) 2. La app pregunta a la API los servidores disponibles para procesar y los recibe 3. La app le pregunta a cada servidor de proceso cuantos procesos tiene en cola 4. La app decide en base a eso a donde enviar el archivo 5. El usuario debe seleccionar un archivo de música a enviar 6. La app envía a la API una petición para que registre un proceso en la base de datos y le regresa el ID 7. La app envía el archivo a un servidor de proceso, junto con el ID de proceso 8. Cuando el servidor de proceso recibe un archivo, este lo pone en espera si es que hay más trabajos en proceso 9. Cuando el servidor de proceso termina un trabajo, este envía una petición a la API, diciendo que ya completo el trabajo y completa el registro. Se envía cuatro URL para descargar las pistas y se guardan en la base de datos.
3. Construcción y/o Programación	Fase 1: Servidor de proceso

	<p>Este servidor está escrito en Python. Se hará uso de un script de Linux para implementarlo, el cual instalará las dependencias en el entorno</p> <p>Fase 2: API</p> <p>Esto es una WEB API, programada en C# bajo .NET Core 3.0. Se implementará en un servicio WEB gratuito de Azure. El deployment se hace automáticamente, cada vez que hay algún cambio en el repositorio en la rama definida en Azure.</p> <p>Fase 3: App de Android</p> <p>Esta fase es la más extensa. La app se desarrollará usando la API versión 22 (mínimo Android 5.0).</p>
4. Pruebas	<p>Cuando termine de desarrollar una característica, proporcionaré a algunos de mis compañeros y amigos un APK.</p> <p>Mi familia estará constantemente probando las funcionalidades.</p>
5. Conclusiones generales	<p>Desarrollar una aplicación que se valga de varios servicios es un proceso complicado y largo; debe hacerse con mucha planeación y cuidado.</p>
6. Conclusiones personales.	<p>Este proyecto es el proyecto más grande que haya empezado hasta ahora.</p>

Introducción

La **música** es el arte consistente de una combinación coherente de sonidos y silencios. Forma parte de nuestras vidas desde tiempos muy antiguos. Casi cualquier sonido agradable de la naturaleza puede ser considerado música. Todos sabemos, en menor o mayor medida, apreciar la música, pero pocos pueden hacerla. Aquellos que hacen música se llaman músicos, quienes con diversos instrumentos o su propia voz generan este bello arte.

Todos conocen a algún músico (sea famoso o no), y sabemos la dedicación que le da a su trabajo, que no es fácil. Muchos de ellos interpretan música de otros artistas, para lo cual necesitan, además de muchos conocimientos y habilidades de música, aprender la melodía y/o letra de canciones.

Los músicos hoy en día cuentan con la posibilidad de tener acceso a audio digital de casi cualquier canción, para poderlas apreciar y aprender. Sin embargo, a veces resulta complicado escuchar claramente instrumentos como el bajo o la propia voz. Una solución a este problema es recurrir a empresas dedicadas a ofrecer pistas instrumentales de muchas canciones. Esto es muy bueno, pero tiene como desventaja que se debe pagar, y además que las pistas no son las originales, sino tocadas por otros músicos para ese fin; incluso pueden ser pistas tocadas con teclado.

Los bajistas podrían apoyarse con filtros *pasa-bajos*, pero eso podría no resultar tan efectivo, pues el bajo no es el único instrumento que genera frecuencias bajas (otros pueden ser: el piano o el bombo de la batería). En fin, estos métodos relacionados con reducir ciertas frecuencias y amplificar otras es un método que no funciona muy bien (nada bien la mayoría de los casos).

Hasta este punto, lo que queda es tener un oído muy agudo, que permita distinguir muy bien los instrumentos. Pero, en los últimos años ha habido grandes avances en el campo de la inteligencia artificial. La inteligencia artificial es posible gracias a diversas técnicas de programación informática, que permiten imitar comportamientos considerados inteligentes. Estos sistemas analizan enormes cantidades de datos, los cuales representan información del mundo real.

La inteligencia artificial ha permitido muchos avances. Desde hace ya algunos años, Facebook analiza las fotografías y trata de predecir la presencia de las personas en dichas fotografías; esto se conoce como *etiquetar* personas. También, tenemos los sistemas autónomos de conducción de algunos autos, que pueden llegar a controlar por completo el vehículo, o al menos detenerlo en caso de ir hacia un obstáculo. Y no podemos dejar fuera de esto a los asistentes digitales (como Alexa, Siri, Cortana o Google Assistant), que para muchos ya es parte de la vida cotidiana.

Pues bien, todos los sistemas mencionados, son capaces de mejorar conforme están en funcionamiento, otros son actualizados manualmente con más y mejores datos colectados. Algo que tienen en común, funcionan con una basta cantidad de información recolectada de muchos lugares.

Una técnica que en este estudio importa mucho es la denominada *Deep learning*, la cual es una técnica de *Machine Learning*, consiste básicamente en proporcionar un conjunto de dato, los cuales consisten en dos partes: el dato de entrada y lo que se espera obtener de él. Se podría entender como aprender en base a ejemplos, pero es más complicado. Un ejemplo clásico: se desea que un programa sea capaz de identificar donde está un gato en una imagen. Inicialmente se basa en analizar las zonas oscuras y claras de la imagen, después busca formas sencillas (como líneas), después pasa al tercer nivel, donde busca formas más complejas (como óvalos). Este proceso continúa hasta que el modelo logra identificar gatos. Este ejemplo tiene relación al hecho de poder identificar caracteres impresos y representarlos como caracteres digitales informáticos.

En este punto empieza a estar claro la relación entre la música y la inteligencia artificial. Los seres humanos podemos de cierto modo concentrar nuestra atención a una conversación entre varias otras, esto es aislar. Los mismo aplica para la música: podemos concentrarnos en un instrumento, aunque esto no es siempre fácil. Es aquí donde es útil el *Deep Learning*: utilizar un modelo que aisle ciertas pistas de la música

Como se mencionó antes, el Deep Learning consiste en tener un conjunto de datos, donde se encuentra el dato de entrada y su salida esperada. En este caso, sería

tener una cierta cantidad de canciones con sus respectivas pistas (bajo, batería, voces y otros). En base a esos datos, la red neuronal es entrenada y puede aislar las pistas de casi cualquier canción (no solo las incluidas en el conjunto de datos).

El principio básico de este proyecto es una aplicación móvil de Android que haga uso de esta tecnología para dicho propósito (aislar las pistas mencionadas). Pero, debido a que este proceso es costoso en términos de procesamiento, el dispositivo generalmente no es capaz de ejecutarlo eficazmente (sea por falta de memoria o capacidad de proceso). Para solucionar dicho problema, se hace uso del modelo cliente-servidor, dotando así a la aplicación de capacidad extra de proceso (servidor).

Aquí, el usuario de la aplicación debe contar con el archivo de audio de la música cuyas pistas desea aislar. Con este archivo, el usuario debe escoger el archivo en la aplicación y este se manda al servidor a procesar; el usuario solo tiene que esperar y el teléfono no utiliza su CPU para este efecto. Cuando el proceso ha sido completado, se le notifica al usuario, y el archivo ya está disponible desde la misma aplicación.

En las siguientes páginas de este documento, se explica más a detalle el funcionamiento de todo el sistema, así como la implementación en software del método de Deep Learning. Se explica a detalle la arquitectura del sistema, el servidor que se utiliza para procesar las canciones y cómo funciona la aplicación y como esta interactúa con las demás partes.

Este proyecto no sería posible sin el modelo **Demucs**, así como su implementación.

Documento de estudio de Demucs:

<https://hal.archives-ouvertes.fr/hal-02379796/document>

Repositorio de código con el código de la implementación de Demucs:

<https://github.com/facebookresearch/demucs>

Antecedentes

Efecto coctel

Cherry fue el primero en notar el efecto coctel (Cherry, 1953): cómo el cerebro es capaz de aislar una conversación de un montón de gente hablando en una habitación. Bregman intentó después entender cómo es que el cerebro puede analizar una señal auditiva compleja y separarla en partes de más alto nivel (Bregman, 1990).

Mezcla y masterización de música

Cuando se produce la música, está se graba por partes, y estas partes se mezclan para formar la canción. Es cómo cuando se produce un automóvil: primero se fabrican sus partes, y luego se ensambla.

La masterización es pulir la mezcla. Generalmente se utilizan técnicas de ecualización, limitación, etc. Es para que la mezcla suene lo mejor posible. La masterización nos ayuda a distinguir mejor los instrumentos de la canción, y, por lo tanto, ayuda a la red neuronal en el proceso de separación.

Separación de orígenes de música en el dominio de forma de onda

A diferencia del efecto coctel, aquí hay más de un origen de interés, caracterizados por diferentes timbres y tonos.

En la evaluación SiSec Mus para separación de música (Stöter et al., 2018), las pistas o tallos se agrupan en 4 categorías: batería, bajo, otros y vocales. El objetivo es generar esas 4 pistas a partir de un origen (llamado mezcla).

Demucs es un modelo de Deep Learning, inspirado por Conv-Tasnet. Demucs opera directamente a nivel de onda de entrada sin procesar y genera una forma de onda para cada fuente. Se ha probado que operar con la forma de onda en lugar del espectrograma es significativamente mejor. También, se hicieron evaluaciones humanas a Demucs y resultó en una mejor percepción.

Servidor HTTP

Un servidor HTTP o servidor WEB, es un programa que se encarga de procesar datos y enviar una respuesta a quien envíe una petición. Tanto la petición como la respuesta tienen una estructura similar. Trabaja bajo el protocolo TCP/IP. El servidor escucha por un puerto de comunicación TCP (generalmente el 80). Existen muchos de estos servidores, y permiten la disponibilidad de todos los sitios web.

Cola de procesos

Es una lista enlazada, que contiene referencias a los procesos que se van a ejecutar, uno por uno, en base a diferentes criterios (pudiendo ser prioridad, complejidad, o el orden en que llegan). Es especialmente útil con procesos que requieren de una gran capacidad de procesamiento y memoria, donde si se hiciera en paralelo, la memoria podría agotarse y hacer caer el sistema. Pueden implementarse variantes donde se ejecuten más de un proceso a la vez, todo depende de la situación.

Objetivos

Desarrollar la aplicación **Isolaatti** tiene como objetivos:

- Permitir a los usuarios utilizar Demucs de manera sencilla, desde la comodidad de una interfaz gráfica en sus dispositivos Android.
- Brindar una interfaz fácil de comprender, siguiendo los estándares de diseño de interfaces gráficas de Android. Esto para facilitar el entendimiento del usuario respecto a la aplicación.
- Brindar la posibilidad de manejar un mezclador sencillo en la aplicación, permitiendo al usuario silenciar o activar las pistas de quiera en tiempo real.
- Ser de utilidad para todo mundo, en especial para los músicos.
- Permitir el uso de la aplicación en diferentes dispositivos que se posean, mediante cuentas de usuario que guardan preferencias y proyectos (canciones con las pistas separadas).
- Permitir compartir los proyectos a otros usuarios, incluso si estos no están registrados.
- Permitir al usuario ecualizar cada pista de manera sencilla.
- Permitir la futura expansión del servicio. Esto quiere decir que es posible desarrollar un cliente WEB, una cliente para iOS, etc., gracias a que se desarrolló una API.

En resumen, la aplicación tiene como objetivo servir de apoyo a los músicos, para que les resulte menos difícil aprender una canción que quieran interpretar con su instrumento.

Metas

Cuando desarrollo Isolaatti tengo cómo metas a largo plazo:

- Hacer llegar la aplicación de Android a una etapa donde el funcionamiento sea perfecto.
- Hacer más robusto todo lo que está detrás de la aplicación.
- Garantizar la disponibilidad del servicio todo el tiempo.
- Mejorar el servicio. Esto incluye disminuir el tiempo de proceso y manejar los errores que puedan salir en el servidor, de manera que no afecte a los archivos de los usuarios.
- Más servidores: para poder procesar en paralelo, lo que se traduce en evitar largas esperas a los usuarios.
- Hacer llegar la aplicación a más usuarios.
- Monetizar la aplicación con anuncios.
- Integrar Google Drive a la aplicación.
- Permitir a los usuarios comunicarse entre sí. Esto limitado a compartir canciones con amigos, pero sin depender de otro servicio de mensajería o red social.
- Agregar la posibilidad de completar los metadatos de las canciones con información de una API de metadatos.
- Permitir localizar la letra de las canciones, directamente desde la aplicación.

Isolaatti busca ayudar a los músicos, pero también a personas curiosas o interesadas en la música. Pero, por la facilidad de uso, cualquier persona podrá usarla sin mayor complicación.

Sustento de la realización del proyecto

Disminución de gastos para los músicos

Muchas veces, las agrupaciones musicales tienen que comprar pistas aisladas, y tales pistas no son baratas. Con Isolaatti, pueden intentar aislar las pistas, y generalmente se obtienen buenos resultados en lo siguiente:

- Aislar batería
- Aislar bajo
- Aislar voces
- Combinaciones como:
 - Quitar voces para generar pistas instrumentales
 - Quitar el bajo
 - Quitar la batería

Lo anterior se puede hacer directo desde la misma aplicación, obteniendo una vista previa de la mezcla.

Apoyando al conocimiento tecnológico

La sociedad está cada vez más consciente de lo que el avance de la inteligencia artificial permite. Isolaatti informa al usuario las imperfecciones del sistema, pero también como funciona a grandes rasgos. Esto contribuye a que el usuario aumente su conocimiento acerca de lo que puede hacer la IA y los sistemas distribuidos y basados en cliente-servidor.

Experiencia de planear un proyecto de software

Planear un proyecto de software, cualquiera que sea, es todo un proceso. Está claro que no se puede ir directo al código, sino que antes se debe concebir todas las partes del sistema.

Experiencia de desarrollo

Este es uno de los primeros proyectos más grandes que desarrollo. Lógicamente, programar, configurar e implementar todas las partes me ha puesto a estudiar diferentes tecnologías, que de no haber hecho un proyecto como este no habría estudiado (al menos por ahora). Desde programar para Android, hasta configurar un servidor en Linux.

Experiencia de proveer un servicio

Definitivamente no es lo mismo hacer que una aplicación local llegue a muchos usuarios y que estos tengan una experiencia agradable, que hacer que una aplicación que depende de múltiples servidores se mantenga en pie y de una experiencia agradable al usuario.

En efecto, proveer un servicio como este requiere dinero. Monetizar la aplicación es una buena forma de obtener ingresos para mantener todo funcionando. Claro que una buena programación es muy importante para aprovechar al máximo los recursos computacionales.

Resumen

Como impacto social clasifico tanto el apoyo a la reducción de gastos para los músicos en concepto de compra de pistas instrumentales, pero también el impacto para los demás usuarios (no músicos) así como el impacto que he experimentado como desarrollador de software.

Desarrollar este proyecto es un paso hacia mi futuro como desarrollador de software profesional.

Metodología de desarrollo

Una de las cosas más difíciles, que se necesitan hacer para hacer que el servicio esté disponible para un número mayor de personas, es utilizar múltiples maquinas corriendo el servidor de proceso.

Una sola maquina con el hardware actual (procesador de doble núcleo y 8GB de RAM), puede procesar una canción a la vez, esto limitado por la memoria. De acuerdo con mediciones tomadas:

$$t(x) = 1.3x$$

Donde $t(x)$ es el tiempo que tarda la maquina y x es el tiempo de duración de la canción en segundos.

A la expresión anterior es necesario sumarle el tiempo necesario para decodificar el audio origen (este proceso se hace antes de comenzar la separación) y el tiempo necesario para codificar las pistas generadas a mp3 (esto se hace después de que se obtienen los archivos .wav).

El tiempo de decodificación es en promedio

$$td(x) = 0.216x$$

El tiempo de codificación a mp3 es despreciable

Entonces, supongamos que una canción dura en promedio 3 minutos. Calculemos cuantas canciones podrían procesarse:

Primero se calcula el tiempo que tardaría una canción de 3 minutos en procesarse

$$x = (3)(60) = 180s$$

$$t(180) = 1.3(180) = 234s$$

$$td(180) = 0.216(180) = 38.88s$$

$$T = t(180) + td(180) = 234s + 38.88s = 272.88s$$

La canción tardaría de manera aproximada 272.88s o 4.548 minutos o

4:32 en notación de reloj

En base a lo anterior:

$$\frac{3600s}{272.88s} = 13.19$$

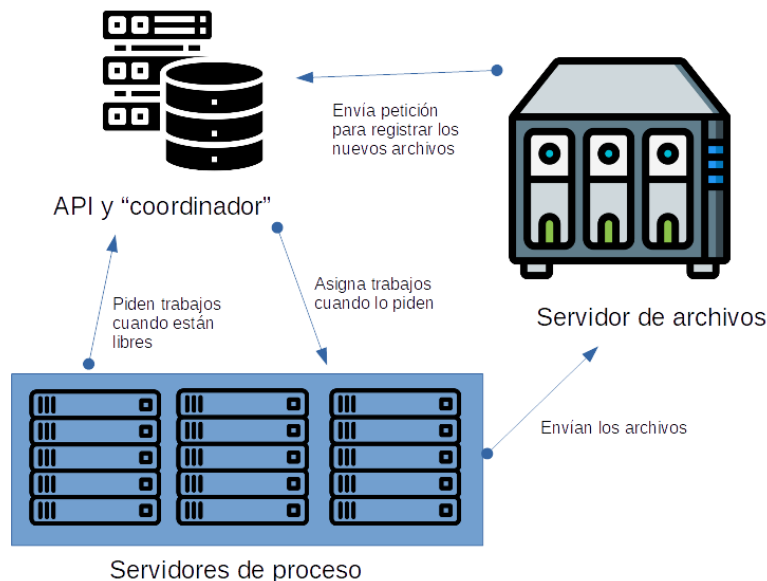
En promedio, se pueden procesar hasta **13 canciones de 3 minutos por hora**.

Una de las metas de Isolaatti (y mías) es poder soportar más usuarios que unos cuantos amigos. En base a los cálculos anteriores, es fácil darse cuenta la cantidad de maquinas iguales que se necesitarían para dar soporte a x cantidad de personas.

Actualmente el sistema cuenta con un servidor que recibe todos los trabajos. Esto podría mejorar si se tienen más maquinas. Para eficientizar las cosas, las maquinas ya no deberían recibir trabajos, sino pedirlos cuando este desocupado; el servidor central debe coordinarlos.

Supongamos que se tienen 10 máquinas, esto permitiría procesar hasta 130 canciones de 3 minutos cada una.

Otro aspecto importante es el almacenamiento. Ahora mismo se deja todo en el único servidor de proceso, por lo que esta máquina también sirve de servidor de archivos. Esto no es buena idea si se tienen múltiples maquinas de proceso, por lo que es mejor tener un único almacenamiento de archivos dedicado a ello.



1 Sistema mejorado (no actual)

Diagrama de Gantt

[illegible]

Productos Entregables

Fases

Programación del script de servidor de proceso

Este está programado en Python. Se hace uso del micro framework web Flask para recibir los archivos y encolar las canciones. Se tiene una cola de procesos, la cual llama a Demucs canción por canción (una a la vez).

Se tiene un hilo que se encarga de ver si hay un nuevo elemento en la cola, y si lo hay, lo manda a procesar. Esta parte es muy importante para que funcione bien el servidor.

Se ha limitado el numero de hilos que puede tomar Demucs a 1, pues la maquina solo tiene dos paralelos, y si se usan los dos, el servidor no podría responder bien a las peticiones, cuando los clientes envían canciones a encolar, dando como resultado tiempos de respuesta muy altos o errores 500 o similares.

Programación de la API

Esta es consumida por la aplicación (cliente). Le permite al cliente crear registros en la base de datos. Además, esta se encarga de enviar las notificaciones a los clientes. En el manual técnico se explica como consumir esta API, especificando los verbos HTTP, las rutas, los parámetros y para qué sirve cada cosa.

Configuración de la base de datos

Entity Framerowk Core (corriendo en la misma máquina que la API) crea las tablas y las relaciones en la base de datos. De este modo, se evita la necesidad de hardcodear queries de SQL, lo cual puede ser inseguro.

Despliegue de la API

Se utiliza un App Service de Azure gratuito. Actualmente aquí está la base de datos (SQLite para pruebas).

Implementación de maquinas virtuales con el script de servidor de proceso

Se está rentando un plan Estándar D2s_v3 (2 Núcleos, 8 GiB de memoria) de Azure. Este está corriendo Ubuntu 18.04 64 bits. Cuenta con 30 GB de almacenamiento.

Toda la configuración de esta maquina ha sido hecha usando SSH. Se configuró el script de servidor como un servicio, para que esté corriendo siempre que la maquina esté encendida.

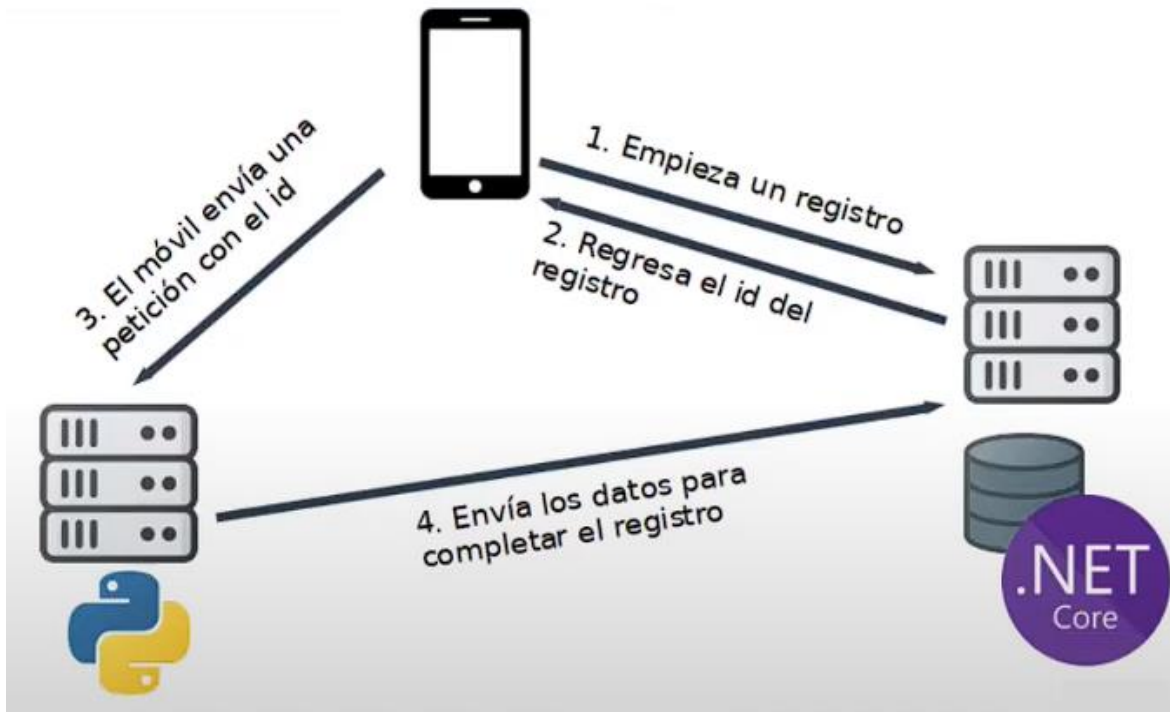
Se ha instalado un servidor web Apache, que hace de servidor de archivos.

Diseño y programación de la aplicación de Android

Esta etapa es la que más tiempo requiere; se necesita manejar muchas cosas relacionadas con Android. La función básica es permitir al usuario subir canciones al servidor para procesarlas y obtener las cuatro pistas. En el manual de usuario se especifica como usarla y en manual técnico se explica cómo funciona internamente.

Arquitectura de la aplicación

Observe el diagrama. La arquitectura actual del sistema está compuesta de tres partes fundamentales. En el diagrama puede observar las interacciones de dichas partes.



Tecnologías utilizadas

Máquina de desarrollo

Se ha utilizado un PC convencional con Ubuntu MATE 18.04 instalado. Un procesador Intel Pentium de cuarta generación con 8 GB de memoria instalada. Es un PC ensamblado por piezas.

Android Studio

Este es el IDE ideal para desarrollar aplicaciones para Android. Se utiliza en Linux para una mejor fluidez, debido a la lentitud con la que se ejecuta en Windows en hardware ajustado.

Control de versiones

Se utiliza git y GitHub para control de versiones del código fuente de los diferentes componentes (aplicación de Android, servidor y API).

Servicio en la nube

Se utiliza Microsoft Azure para todos los servicios de Hosting. Esto incluye el App Service gratuito con una instancia de MySQL (la cual no es utilizada por ahora), y una máquina virtual corriendo Linux.

Cuenta de correo electrónico de Google

Se utiliza una cuenta de Gmail para enviar los correos de verificación de cuenta, así como avisos a los usuarios. Esta cuenta es una cuenta personal, creada para este propósito. Mientras siga en fase de pruebas y desarrollo no es necesario usar otro servicio de correo electrónico.

Google Firebase Cloud Messaging

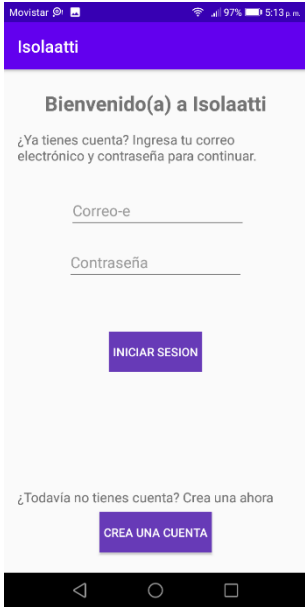
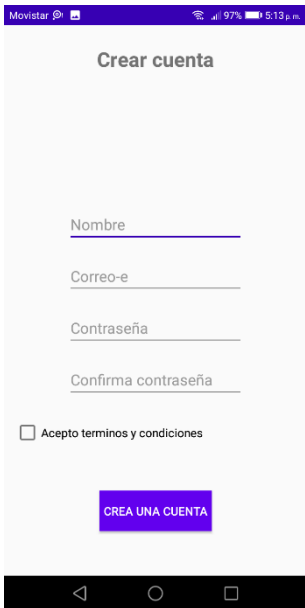
Se utiliza este servicio para enviar notificaciones a los usuarios. Basta con contar con el token de Google del dispositivo. Cabe destacar que el dispositivo Android del usuario debe contar con los servicios de Google instalados para poder utilizar esto.

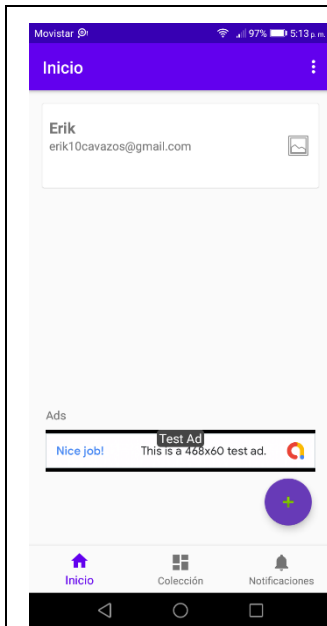
Lenguajes utilizados

Se utilizan los siguientes lenguajes de programación:

1. Kotlin: para el desarrollo de la aplicación de Android
2. C#: para el desarrollo de la API. Compila bajo .NET Core 3.1
3. Python: para el servidor de proceso. Se utiliza Python 3.7
4. HTML5 (HTML, CSS y JavaScript): en menor medida, para el desarrollo web necesario para que funcionen los links de compartimiento (para enviar a otras personas).

Diseño de pantallas

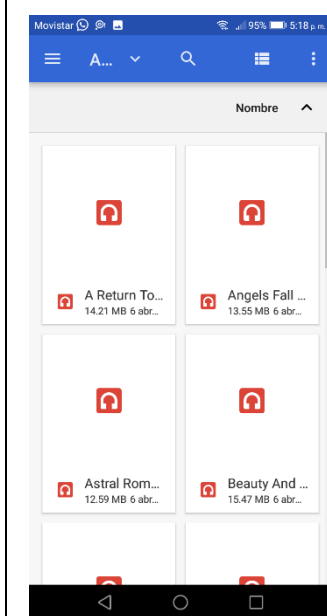
Pantalla	Descripción breve
	Pantalla de inicio de sesión Aquí el usuario debe introducir sus credenciales. De no tener una cuenta el usuario debe tocar el botón “CREA UNA CUENTA”.
	Pantalla de crear cuenta Aquí el usuario debe introducir los datos solicitados para crear una cuenta. Después, se le envía un correo a la dirección que se proporcionó, esto con el fin de verificarla. Si el usuario no verifica la cuenta, esta cuenta no funcionará.



Pantalla de inicio

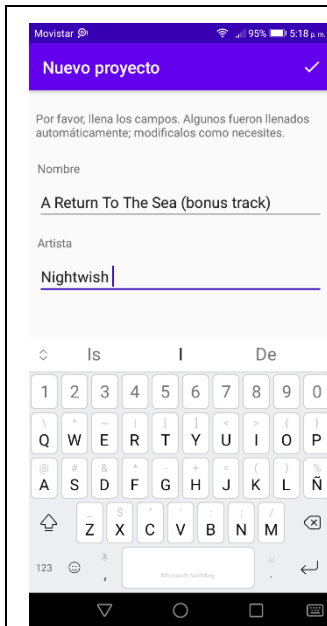
Aquí es donde el usuario llega cuando inicia la aplicación desde el launcher. Únicamente se muestra la información de su perfil y un botón para ir a la pantalla de carga de archivo.

NOTA: Al tocar el cuadro donde viene la información de perfil se accede a la pantalla de perfil, desde donde se puede cambiar la contraseña y otras cosas.



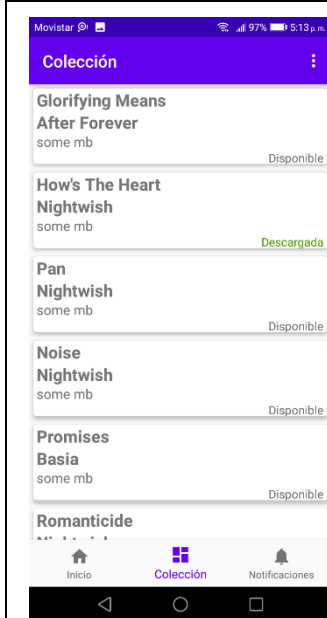
Pantalla de carga de archivos

Esta pantalla es proporcionada por el teléfono, por lo que el diseño puede variar. Aquí es donde el usuario escoge qué canción quiere subir para hacer un proyecto de separación.



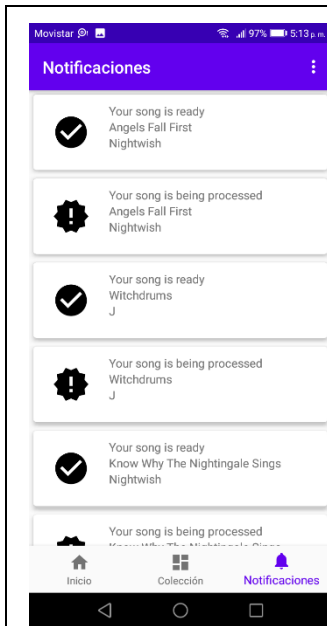
Pantalla de información de nuevo proyecto

Esta pantalla es presentada después de que el usuario escoge su archivo. Aquí el usuario debe asegurarse de que los datos de la canción se encuentran como quiere que aparezcan en su colección.



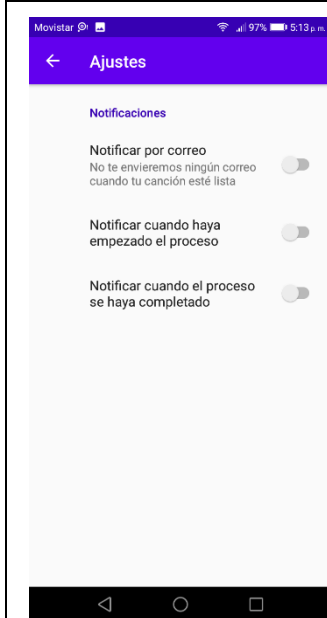
Pantalla de colección

Aquí se muestran las canciones que el usuario tiene en su colección. Desde aquí se pueden eliminar, editar metadatos, y abrirlas.



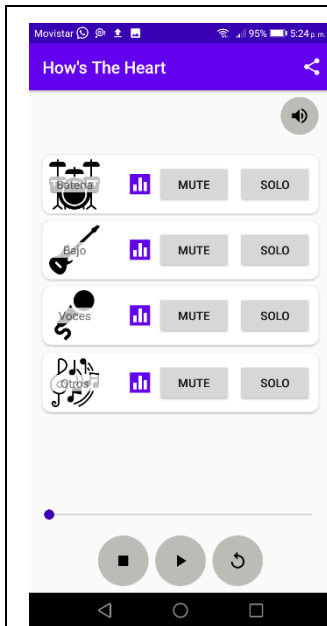
Pantalla de colección

Aquí se muestra el historial de notificaciones que le han llegado al usuario, así como las que fueron enviadas, pero no recibidas en su momento.



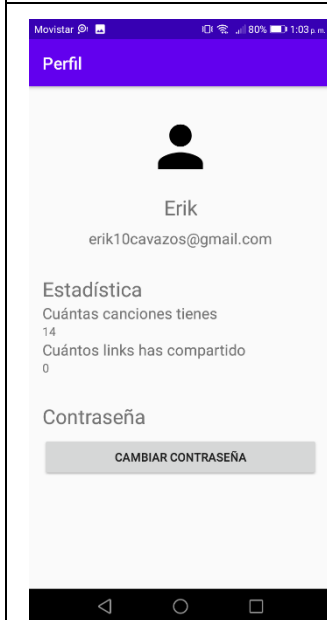
Pantalla de ajustes

Aquí se concentran los ajustes de la aplicación. Por el momento, solo se pueden controlar las notificaciones.



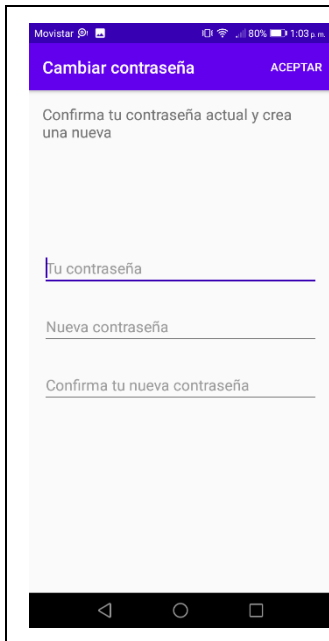
Pantalla de reproductor de pistas

Esta pantalla es la más importante de la aplicación. Desde aquí, el usuario puede generar las combinaciones que quiera con las pistas. Además, cuenta con un ecualizador sencillo por pista.



Pantalla de perfil

Aquí el usuario puede ver su nombre, correo y un conteo de canciones y links. También, desde aquí puede cambiar la contraseña.



Pantalla de cambio de contraseña

Esta pantalla sirve para cambiar la contraseña. Si el proceso se hace correctamente, se cerrará sesión automáticamente y se le pedirá al usuario volver a iniciar sesión con su nueva contraseña.

Implementación

Todos los servidores se encuentran escuchando en Internet. Se ha utilizado los servicios en la nube de Azure y Google Firebase. Realmente la aplicación no está disponible aun para todo usuario; está se encuentra, sin embargo, publica en su propio repositorio de GitHub. En este repositorio se suben los APKs solamente.

Módulos

La aplicación está compuesta de los siguientes módulos:

1. Módulo general
 - a. Comenzar nuevo proyecto
 - i. Selección de archivo
 - ii. Introducción de datos
 - b. Cambiar información de cuenta
 - i. Cambio de contraseña
 - ii. Cambio de nombre de usuario
2. Módulo de canciones
 - a. Colección
 - i. Eliminar canción
 - ii. Modificar información de canción
 - b. Reproductor de pistas
 - i. Ecualizador
 - ii. Compartir
3. Módulo de notificaciones
4. Módulo de ajustes
5. Módulo de cuentas
 - a. Creación de cuenta
 - b. Iniciar sesión

Librerías

En la aplicación se utilizan las siguientes librerías:

- Volley: librería para hacer peticiones HTTP a servidores.
<https://developer.android.com/training/volley>
- Android Upload Service: para subir archivos por POST.
<https://github.com/gotev/android-upload-service>

En el lado de la API se utilizan las siguientes librerías y SDKs

- El SDK de admin de Firebase para C#: Para poder enviar notificaciones a los dispositivos desde el servidor.
<https://firebase.google.com/docs/admin/setup?hl=es#c>

En el servidor de proceso se utiliza el siguiente micro-framework:

- Flask: para manejar las peticiones HTTP.
<https://flask.palletsprojects.com/en/1.1.x/>

Scripts

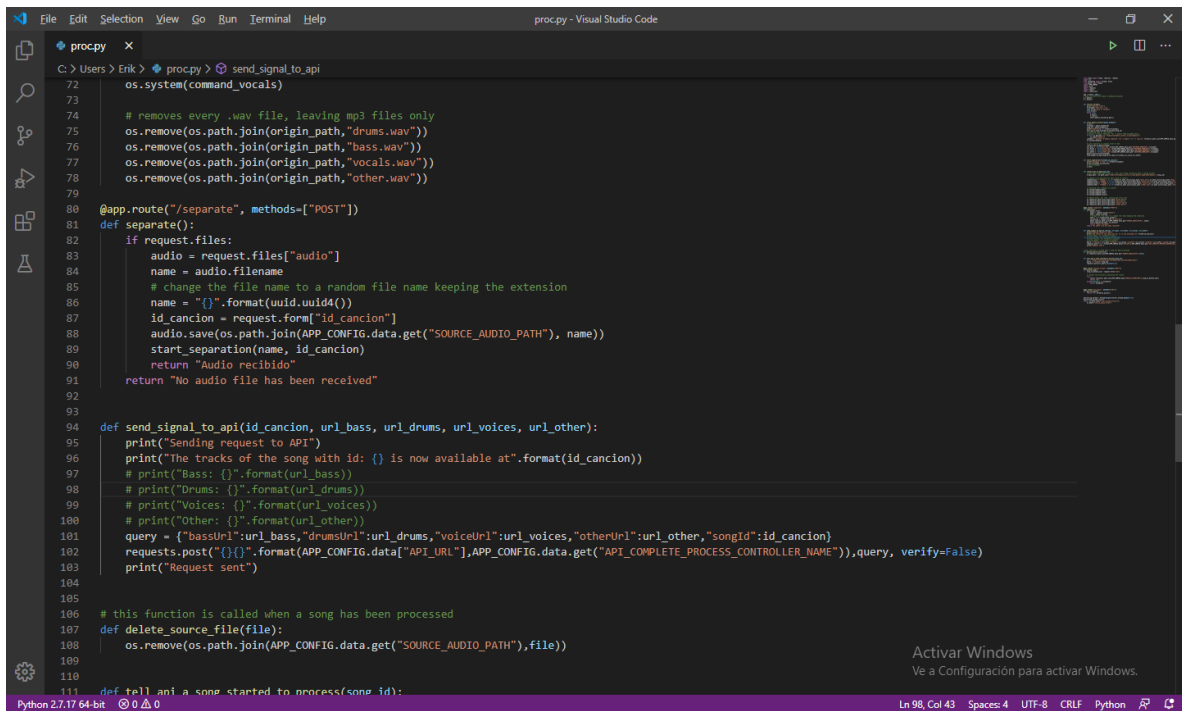
A continuación, proporciono una muestra del código del script de Python del servidor de procesos.

```
File Edit Selection View Go Run Terminal Help
proc.py - Visual Studio Code

proc.py x
C:\Users\Enik> proc.py
1 from flask import Flask, redirect, request
2 import os
3 from threading import Thread, Event
4 from queue import Queue
5 import APP_CONFIG
6 import uuid
7 import requests
8 import shutil
9 import subprocess
10
11 app = Flask(__name__)
12 # queue to store file names of pending processes
13 q = Queue()
14 e = Event()
15
16
17 def starter_thread():
18     print("Waiting files")
19     file=open('out2.txt','w')
20     file.write("jaja el thread")
21     file.close()
22     while True:
23         e.wait()
24         e.clear()
25         start_demucs_process(q.get())
26
27
28 def start_demucs_process(queue_element):
29     e.set()
30     filename = queue_element[0]
31     song_id = queue_element[1]
32     print("Separating {}".format(filename))
33     tell_api_a_song_started_to_process(song_id)
34     # executing this on the console
35     # python3 -m demucs.separate --dl -n demucs PATH_TO_AUDIO_FILE_1
36     if (not os.getcwd() == "/home/erik/demucs_server_script/demucs"):
37         os.chdir("demucs")
38     command = 'python3 -m demucs.separate --dl -n demucs "{}" >> log.txt'.format(os.path.join(APP_CONFIG.data.get("SOURCE_AUDIO_PATH"), filename))
39     os.system(command)
```

```
File Edit Selection View Go Run Terminal Help
proc.py - Visual Studio Code

proc.py x
C:\Users\Enik> proc.py
35 # python3 -m demucs.separate --dl -n demucs PATH_TO_AUDIO_FILE_1
36 if (not os.getcwd() == "/home/erik/demucs_server_script/demucs"):
37     os.chdir("demucs")
38 command = 'python3 -m demucs.separate --dl -n demucs "{}" >> log.txt'.format(os.path.join(APP_CONFIG.data.get("SOURCE_AUDIO_PATH"), filename))
39 os.system(command)
40
41 # calls function to convert files to mp3
42 convert_wav_to_mp3(filename)
43 url_bass = '{}({})/bass.mp3'.format(APP_CONFIG.data.get("HOSTNAME_DOWNLOAD"),filename)
44 url_drums = '{}({})/drums.mp3'.format(APP_CONFIG.data.get("HOSTNAME_DOWNLOAD"),filename)
45 url_voice = '{}({})/vocals.mp3'.format(APP_CONFIG.data.get("HOSTNAME_DOWNLOAD"),filename)
46 url_other = '{}({})/other.mp3'.format(APP_CONFIG.data.get("HOSTNAME_DOWNLOAD"),filename)
47 delete_source_file(filename)
48 send_signal_to_api(song_id,url_bass,url_drums,url_voice,url_other)
49
50
51 def start_separation(filename,id_cancion):
52     print("Agregado a cola: {}".format(filename))
53     q.put([filename,id_cancion])
54     #q.put(filename)
55     e.set()
56
57
58 def convert_wav_to_mp3(song_uid):
59     # path where separated tracks in ".wav" are stored (directory that is being served)
60     origin_path = os.path.join("/home/erik/demucs_server_script/demucs/separated/demucs/",song_uid)
61
62     # command to be executed for each track to convert
63     command_bass = "ffmpeg -i {} {}".format(os.path.join(origin_path,"bass.wav"),os.path.join(origin_path,"bass.mp3"))
64     command_drums = "ffmpeg -i {} {}".format(os.path.join(origin_path,"drums.wav"),os.path.join(origin_path,"drums.mp3"))
65     command_vocals = "ffmpeg -i {} {}".format(os.path.join(origin_path,"vocals.wav"),os.path.join(origin_path,"vocals.mp3"))
66     command_other = "ffmpeg -i {} {}".format(os.path.join(origin_path,"other.wav"),os.path.join(origin_path,"other.mp3"))
67
68     # executing four commands to convert
69     os.system(command_bass)
70     os.system(command_drums)
71     os.system(command_other)
72     os.system(command_vocals)
73
74     # removes every .wav file, leaving mp3 files only
```



```
72 os.system(command_vocals)
73
74 # removes every .wav file, leaving mp3 files only
75 os.remove(os.path.join(origin_path, "drums.wav"))
76 os.remove(os.path.join(origin_path, "bass.wav"))
77 os.remove(os.path.join(origin_path, "vocals.wav"))
78 os.remove(os.path.join(origin_path, "other.wav"))
79
80 @app.route("/separate", methods=["POST"])
81 def separate():
82     if request.files:
83         audio = request.files["audio"]
84         name = audio.filename
85         # change the file name to a random file name keeping the extension
86         name = "{}".format(uuid.uuid4())
87         id_cancion = request.form["id_cancion"]
88         audio.save(os.path.join(APP_CONFIG.data.get("SOURCE_AUDIO_PATH"), name))
89         start_separation(name, id_cancion)
90         return "Audio recibido"
91     return "No audio file has been received"
92
93
94 def send_signal_to_api(id_cancion, url_bass, url_drums, url_vocals, url_other):
95     print("Sending request to API")
96     print("The tracks of the song with id: {} is now available at".format(id_cancion))
97     # print("Bass: {}".format(url_bass))
98     # print("Drums: {}".format(url_drums))
99     # print("Voices: {}".format(url_vocals))
100    # print("Other: {}".format(url_other))
101    query = ("bassUrl":url_bass,"drumsUrl":url_drums,"voiceUrl":url_vocals,"otherUrl":url_other,"songId":id_cancion)
102    requests.post("{}{}".format(APP_CONFIG.data["API_URL"],APP_CONFIG.data.get("API_COMPLETE_PROCESS_CONTROLLER_NAME")),query, verify=False)
103    print("Request sent")
104
105
106 # this function is called when a song has been processed
107 def delete_source_file(file):
108     os.remove(os.path.join(APP_CONFIG.data.get("SOURCE_AUDIO_PATH"),file))
109
110
111 def tell_api_a_song_started_to_process(song_id):
```

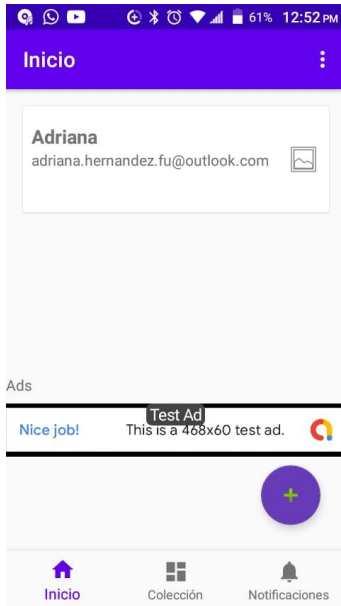
Código demostrativo disponible para visualizar en:

<https://drive.google.com/file/d/1ZZUN1HITtcmTu2jNRidTCRNZuawRcADX/view?usp=sharing>

Pruebas en el cliente

La aplicación de Android se probó en los siguientes teléfonos:

ZTE Blade L7A



Versión de Android: 7.0 Nougat

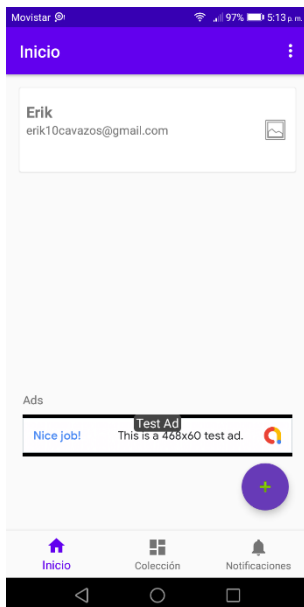
RAM: 1 GB

Almacenamiento: 8GB

Pantalla: 5 pulgadas, 480 x 854

Procesador: Spreadtrum SC7731 Quad Core @ 1.2GHz

Huawei Honor 7S



Versión de Android: 8.1.0 Oreo

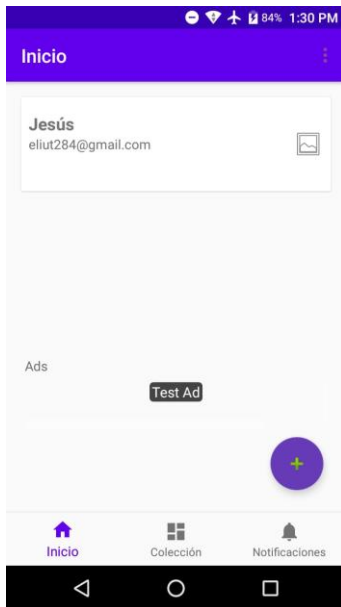
RAM: 2GB

Almacenamiento: 16GB

Pantalla: 5.45 pulgadas, 720 x 1440

Procesador: Mediatek MT6739 Quad Core @ 1.5GHz

M4 SS4458



Versión de Android: 6.0.1 Marshmallow

RAM: 2GB

Almacenamiento: 16GB

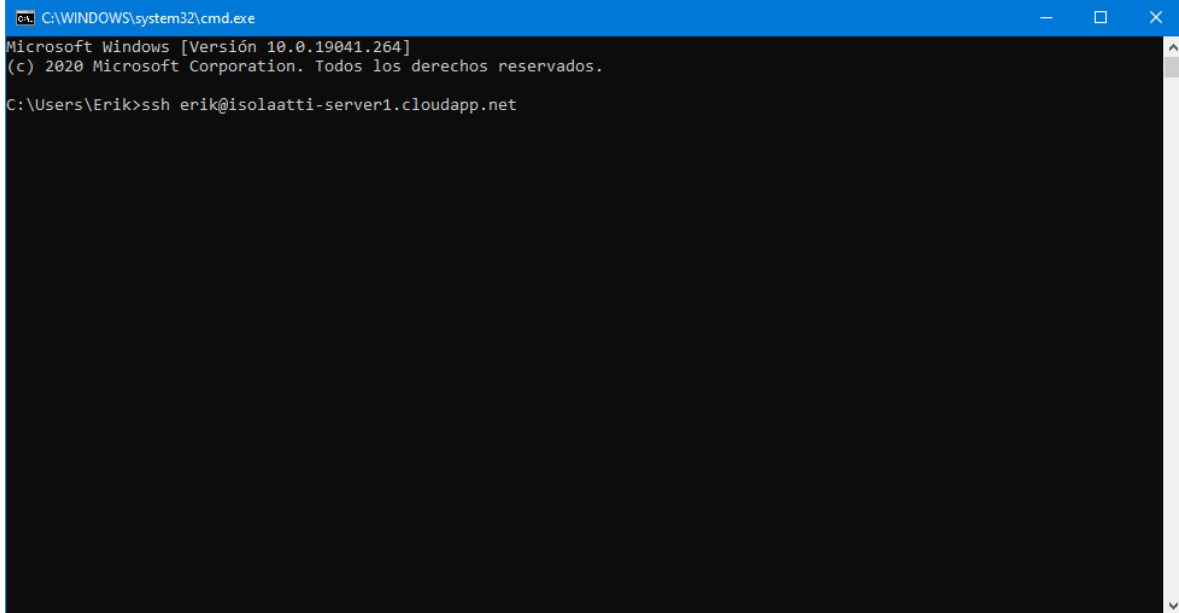
Pantalla: 5.5 pulgadas, HD (720 x 1280)

Procesador: Qualcomm Snapdragon 212 Quad Core @ 1.3GHz

Pruebas en el servidor

El servidor de proceso de canciones (el que corre en Python), actualmente está escuchando en: `isolaatti-server1.cloudapp.net`. En el manual técnico se especifica como usar las rutas de este servidor web.

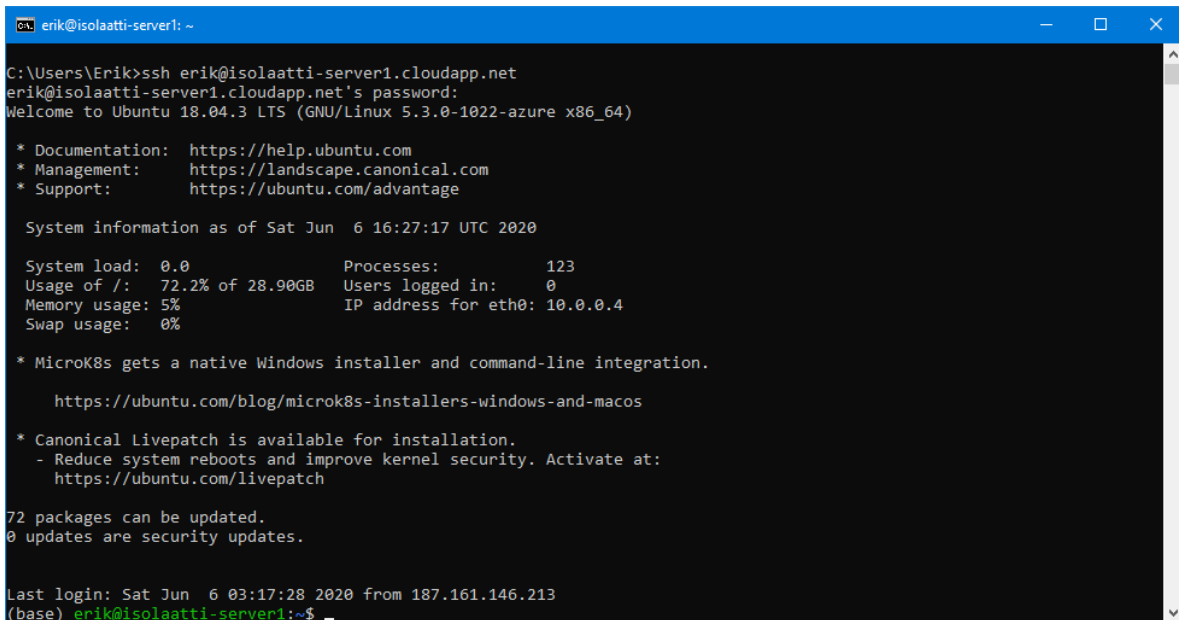
Así se inicia sesión en el servidor por SSH



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Versión 10.0.19041.264]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Erik>ssh erik@isolaatti-server1.cloudapp.net
```

Aquí está la sesión



```
erik@isolaatti-server1: ~
C:\Users\Erik>ssh erik@isolaatti-server1.cloudapp.net
erik@isolaatti-server1.cloudapp.net's password:
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 5.3.0-1022-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Sat Jun  6 16:27:17 UTC 2020

System load:  0.0               Processes:    123
Usage of /:   72.2% of 28.90GB   Users logged in:  0
Memory usage: 5%               IP address for eth0: 10.0.0.4
Swap usage:  0%

 * MicroK8s gets a native Windows installer and command-line integration.

   https://ubuntu.com/blog/microk8s-installers-windows-and-macos

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

72 packages can be updated.
0 updates are security updates.

Last login: Sat Jun  6 03:17:28 2020 from 187.161.146.213
(base) erik@isolaatti-server1:~$
```

Usando el programa htop para ver los procesos. El proceso remarcado es uno de los subprocesos del servidor.

```

erik@isolaatti-server1: ~
1  [|||] 0.8%] Tasks: 40, 69 thr; 1 running
2  [|||] 0.8%] Load average: 0.07 0.02 0.00
Mem [|||||||||||||||||] 302M/7.75G Uptime: 16:32:46
Swp[|||||] 0K/0K

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
1317 syslog 20 0 261M 4944 3360 S 0.0 0.1 0:01.49 /usr/sbin/rsyslogd -n
1274 syslog 20 0 261M 4944 3360 S 0.0 0.1 0:03.17 /usr/sbin/rsyslogd -n
1297 root 20 0 230M 2208 1480 S 0.0 0.0 0:00.12 /usr/bin/lxcfs /var/lib/lxcfs/
1298 root 20 0 230M 2208 1480 S 0.0 0.0 0:00.13 /usr/bin/lxcfs /var/lib/lxcfs/
26132 root 20 0 230M 2208 1480 S 0.0 0.0 0:00.12 /usr/bin/lxcfs /var/lib/lxcfs/
1275 root 20 0 230M 2208 1480 S 0.0 0.0 0:00.40 /usr/bin/lxcfs /var/lib/lxcfs/
1278 daemon 20 0 28328 2108 1900 S 0.0 0.0 0:00.00 /usr/sbin/atd -f
1279 root 20 0 72800 16012 4016 S 0.0 0.2 0:00.21 /usr/bin/python3 -u /usr/sbin/waagent -daemon
1283 erik 20 0 41920 22256 8104 S 0.0 0.3 0:07.32 /home/erik/anaconda3/envs/servidor/bin/python /home/erik
1294 root 20 0 107M 1944 1728 S 0.0 0.0 0:00.00 /usr/sbin/irqbalance --foreground
1284 root 20 0 107M 1944 1728 S 0.0 0.0 0:01.82 /usr/sbin/irqbalance --foreground
1312 root 20 0 281M 5952 5080 S 0.0 0.1 0:03.45 /usr/lib/accountsservice/accounts-daemon
1319 root 20 0 281M 5952 5080 S 0.0 0.1 0:00.00 /usr/lib/accountsservice/accounts-daemon
1286 root 20 0 281M 5952 5080 S 0.0 0.1 0:03.49 /usr/lib/accountsservice/accounts-daemon
1390 root 20 0 166M 13608 5828 S 0.0 0.2 0:00.00 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-star
1288 root 20 0 166M 13608 5828 S 0.0 0.2 0:00.10 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-star
1305 root 20 0 16408 1792 1640 S 0.0 0.0 0:00.00 /sbin/agetty -o -p -- \u --keep-baud 115200,38400,9600 t
1310 root 20 0 14884 1516 1384 S 0.0 0.0 0:00.00 /sbin/agetty -o -p -- \u --noclear tty1 linux
1467 root 20 0 183M 15948 8060 S 0.0 0.2 0:00.00 /usr/bin/python3 /usr/share/unattended-upgrades/unattend
1318 root 20 0 183M 15948 8060 S 0.0 0.2 0:00.09 /usr/bin/python3 /usr/share/unattended-upgrades/unattend
1321 root 20 0 33164 832 0 S 0.0 0.0 0:00.00 nginx: master process /usr/sbin/nginx -g daemon on; mast
1322 noboby 20 0 37940 3452 2168 S 0.0 0.0 0:00.44 nginx: worker process
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice F8Nice F9Kill F10Quit

```

El proceso con PID 64680 de la captura de pantalla es demucs ejecutándose. Obsérvese que usa el 100% de uno de los núcleos y una gran cantidad de la memoria disponible.

```

erik@isolaati-server1: ~
1 [|||||||||||||||||||||||||||||||||||||100.0%] Tasks: 40, 70 thr; 2 running
2 [|||||1.3%] Load average: 0.45 0.11 0.04
Mem[|||||||||||||||||3.85G/7.75G] Uptime: 16:35:47
Swp[|||||0K/0K]

  PID USER      PRI  NI  VIRT   RES   SHR  S  CPU% MEM%   TIME+  Command
64680 erik        20   0 5266M 3751M 114M  R 100.4 47.3   0:21.65 python3 -m demucs.separate --dl -n demucs /home/erik/dem
1481 root         20   0 234M 25952 8220  S  0.7  0.3   4:25.89 python3 -u bin/WALinuxAgent-2.2.48.1-py2.7.egg --run-exth
64455 erik        20   0 32612 4720  3520  R  0.0  0.1   0:00.81 htop
1613 root         20   0 234M 25952 8220  S  0.0  0.3   0:55.19 python3 -u bin/WALinuxAgent-2.2.48.1-py2.7.egg --run-exth
1297 root         20   0 229M 2640  1480  S  0.0  0.0   0:00.13 /usr/bin/lxcfs /var/lib/lxc/fs/
1819 erik        20   0 139M 32748 6224  S  0.0  0.4   0:00.17 /home/erik/anaconda3/envs/servidor/bin/python /home/erik
1718 erik        20   0 139M 32748 6224  S  0.0  0.4   0:02.33 /home/erik/anaconda3/envs/servidor/bin/python /home/erik
1322 nobody     20   0 37940 3328  2044  S  0.0  0.0   0:00.68 nginx: worker process
985 systemd-r  20   0 70744 5680 4992  S  0.0  0.1   0:08.98 /lib/systemd/systemd-resolved
582 root         20   0 12016 2668  1772  S  0.0  0.0   0:27.21 /usr/lib/linu-tools/5.3.0-1022-azure/hv_kvpsd_daemon -n
1609 root         20   0 234M 25952 8220  S  0.0  0.3   0:10.16 python3 -u bin/WALinuxAgent-2.2.48.1-py2.7.egg --run-exth
509 root         19  -1 119M 40764 37532  S  0.0  0.5   0:11.68 /lib/systemd/systemd-journald
64344 erik        20   0 105M 5508 4504  S  0.0  0.1   0:00.07 sshd: erik@pts/0
1286 root         20   0 281M 5016 4144  S  0.0  0.1   0:03.50 /usr/lib/accountsservice/accounts-daemon
1283 erik        20   0 41920 19756 5604  S  0.0  0.2   0:07.33 /home/erik/anaconda3/envs/servidor/bin/python /home/erik
1405 root         20   0 93120 7688 5880  S  0.0  0.1   0:02.37 /usr/sbin/apache2 -k start
1312 root         20   0 281M 5016 4144  S  0.0  0.1   0:03.47 /usr/lib/accountsservice/accounts-daemon
1 root         20   0 78084 8860 6348  S  0.0  0.1   0:04.07 /sbin/init
1315 syslog     20   0 261M 4528 2944  S  0.0  0.1   0:00.84 /usr/sbin/rsyslogd -n
1274 syslog     20   0 261M 4528 2944  S  0.0  0.1   0:03.18 /usr/sbin/rsyslogd -n
1391 root         20   0 72296 5824 5072  S  0.0  0.1   0:02.00 /usr/sbin/sshd -D
1284 root         20   0 107M 1724 1508  S  0.0  0.0   0:01.83 /usr/sbin/irqbalance --foreground

F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice F8Nice F9Kill F10Quit

```

Referencias

(s.f.). Obtenido de <http://xihuitl.mx.tripod.com/coladeprocesos.htm>

Alexandre Défossez, N. U. (25 de Noviembre de 2019). *HAL*. Obtenido de <https://hal.archives-ouvertes.fr/hal-02379796/document>

LANDR. (3 de Junio de 2020). *LANDR*. Obtenido de <https://www.landr.com/es/que-es-la-masterizacion>

Moreno, C. G. (s.f.). Obtenido de Indra Company: <https://www.indracompany.com/es/blogneo/deep-learning-sirve>

neo.lcc.uma.es. (s.f.). *neo.lcc.uma.es*. Obtenido de <http://neo.lcc.uma.es/evirtual/cdd/tutorial/aplicacion/http.html>

Resumen

Isolaatti es una aplicación de Android que usa la red neuronal demucs (de Facebook research) para aislar la batería, el bajo, las voces y otros de una canción. Dicha red neuronal se encuentra detrás de un servidor web, el cual se encarga de recibir los archivos de música y los manda a procesar a la red neuronal uno por uno.

En la aplicación, se cuenta con cuentas de usuario, colección de canciones y lo más importante, un reproductor de pistas, el cual permite escuchar las cuatro pistas al mismo tiempo y silenciar y ecualizar las que se deseen, generando la mezcla que se desee en tiempo real.

La aplicación envía los archivos de música al servidor de proceso mediante HTTP, usando el método POST. Las pistas obtenidas en la red neuronal se ponen a disposición del usuario para su descarga en la aplicación. La aplicación los descarga y permite al usuario escucharlas desde ahí.