



Universidad Autónoma de Nuevo León  
Facultad de Ingeniería Mecánica y Eléctrica



**Isolaatti**  
**Android Application**  
**Documentation**  
**Mobile Devices Engineering**  
**(Ingeniería de Dispositivos Móviles)**  
**Ing. Héctor Hugo Flores Moreno**  
**Thursday N1-N3**

Student: Erik Everardo Cavazos Hernández

Number ID: 1811290

Semester: January – June 2020

## A. PROJECT IDENTIFICATION

Date of Preparation: 13/02/2020

1. Project Name:	<b>Isolaatti:</b> aísla los instrumentos de una canción
2. Learning Unit(s):	Ingeniería de Dispositivos Móviles
3. Responsible Teacher(s):	Ing. Héctor Hugo Flores Moreno
4. Company Name:	
5. Student:	Erik Everardo Cavazos Hernández
6. Teachers linked to the project:	

## B. DESCRIPCIÓN DEL PROYECTO

1. Project Overview:	An application that separates the instruments from a song, so that it can be heard in isolation: drums, bass, vocals, and others (it can be guitars, pianos, synthesizers, metals, etc.)
2. Project objective(s)	<ul style="list-style-type: none"><li>• Make the app usable by anyone without difficulty.</li><li>• Fulfilling the main function</li></ul>
3. Project purpose	<ul style="list-style-type: none"><li>• Reach Android users and be able to earn revenue (with ads), to be able to maintain the hosting of the servers and not lose money at least.</li></ul>
4. Development Tools and/or Materials	<p><b>IDEs and code editors:</b></p> <ul style="list-style-type: none"><li>• Android Studio</li><li>• Visual Studio Code</li><li>• JetBrains Rider</li></ul> <p><b>Programming languages:</b></p> <ul style="list-style-type: none"><li>• Kotlin (Android app)</li><li>• Python (process server script)</li><li>• C# (WEB API)</li></ul> <p><b>Database</b></p>

	<ul style="list-style-type: none"> <li>• MySQL</li> </ul> <b>External software</b> <ul style="list-style-type: none"> <li>• <a href="https://github.com/facebookresearch/demucs">https://github.com/facebookresearch/demucs</a></li> </ul> <b>Libraries, frameworks and SDKs</b> <ul style="list-style-type: none"> <li>• Flask (Python)</li> <li>• .NET Core 3.1</li> <li>• Android SDK</li> </ul> <b>Version control</b> <ul style="list-style-type: none"> <li>• Git (GitHub, remote)</li> </ul> <b>Other</b> <ul style="list-style-type: none"> <li>• Terminal</li> <li>• SSH</li> </ul>
5. Estimate the cost of the project	<p>The development process will not cost money, however, implementing it will.</p> <p>The cost varies depending on the number of compute servers and capacity.</p> <p>Example:</p> <p>A virtual machine with 2 cores and 8 GB of RAM: \$85 per month.</p> <p>Multiple <i>Azure App Services</i> will be used for testing purposes, which are free, but with less throughput.</p> <p>During development, simply use a PC</p>
6. References from previous projects	<p>The project would not be possible without:</p> <p><a href="https://github.com/facebookresearch/demucs">https://github.com/facebookresearch/demucs</a></p> <p>Which works only via command line. This project is deployed on a WEB server.</p>
7. Innovation in the project	<p>Create an Android client with an easy-to-use graphical interface, which makes it easy to get the tracks of a song.</p>

### C. PLANNING ACTIVITIES

Activities	Dates
Write the script in Python that will take care of receiving the audio files and process them to get all four tracks.	February 13
API development	
Database	February 20
Account registration	February 20
Inicio de sesión	February 20
Controller to receive requests from the process server, which will indicate when a file has already been processed, thereby saving the URL of each track in the database, as well as sending a notification to the client (by the Android notification system, and by email)	February 20
Method to send email to the user when their file is ready	February 27
API deployment	February 27
Cloud implementation of the script mentioned in the first activity.	February 27
Android app development	

Start the Android Studio project and set parameters	February 27
Account registration and login screen	March 5
File upload screen	March 5
SQLite database for saving preferences and other things locally	March 12
Screen of processed songs	March 12 and 19
Background process (the one that launches the notification when a standby file has been processed and is ready to be downloaded)	March 26 and April 2
Player to be able to play the songs and mute or not the tracks you want	April 2 and 23
Sending notifications method	May 7
Testing with users	May 14
Bug fixes	May 14 and 21
Ready tuning for the Play Store	May 26

## D. PROJECT DEVELOPMENT

1. Research and/or Analysis of the Project	My application will make use of the client-server model. It is based on one or more waiting lists.
2. Project Design	<p>The project consists of developing a system, composed of three major parts:</p> <ul style="list-style-type: none"><li>• The Android client</li><li>• A WEB API</li><li>• N process servers</li></ul> <p>Summary</p> <ol style="list-style-type: none"><li>1. Sign in (first time only)</li><li>2. The app asks the API for the servers available to process and receives them</li><li>3. The app asks each process server how many processes it has queued</li><li>4. The app decides based on that where to send the file</li><li>5. The user must select a music file to send</li><li>6. The app sends the API a request to register a process in the database and returns the ID</li><li>7. The app sends the file to a process server, along with the process ID</li><li>8. When the process server receives a file, it puts it on hold if more jobs are in progress.</li><li>9. When the process server finishes a job, it sends a request to the API, saying that it already completes the job and completes the logging. Four URLs are sent to download the tracks and saved to the database.</li></ol>
3. Construction and/or Programming	<p><b>Phase 1: Process server</b></p> <p>This server is written in Python. A Linux script will be used to deploy it, which will install dependencies in the environment</p> <p><b>Step 2: API</b></p> <p>This is a WEB API, programmed under the .NET Core 3.0. It will be deployed in a free Azure WEB service. The deployment is done automatically,</p>

	<p>whenever there are any changes to the repository in the branch defined in Azure.</p> <p><b>Phase 3: Android App</b></p> <p>This phase is the most extensive. The app will be developed using API version 22 (minimum Android 5.0).</p>
4. Tests	<p>When I finish developing a feature, I'll provide some of my colleagues and friends with an APK.</p> <p>My family will be constantly testing the features.</p>
5. General conclusions	<p>Developing an application that takes on multiple services is a complicated and lengthy process; needs to be done with a lot of planning and care.</p>
6. Personal conclusions.	<p>This project is the largest project I've ever started.</p>

## Introduction

**Music** is the consistent art of a coherent combination of sounds and silences. It's been part of our lives since ancient times. Almost any pleasant sound of nature can be considered music. We all know, to a lesser or greater extent, to appreciate the music, but few people can make it. Those who make music are called musicians, who with various instruments or their own voice generate this beautiful art.

Everyone knows some musician (whether famous or not), and we know the dedication he gives to his work, which is not easy. Many of them perform music from other artists, for which they need, in addition to many knowledge and music skills, to learn the melody and/or lyrics of songs.

Musicians nowadays have the possibility to have access to digital audio of almost any song, to be able to appreciate and learn them. However, it is sometimes difficult to clearly hear instruments such as bass or voice itself. One solution to this problem is to turn to companies dedicated to offering instrumental tracks of many songs. This is very good, but it has as a disadvantage that you have to pay, and besides that the tracks are not the original ones, but played by other musicians for that purpose; they can even be keyboard-played tracks.

Bass players could lean on *low-pass* filters, but that might not be as effective, as bass is not the only instrument that generates low frequencies (others may be the piano or the drums). In summary, these methods related to reducing certain frequencies and amplifying others is a method that does not work very well (nothing good most cases).

Up to this point, what remains is to have a very sharp ear, which allows to distinguish the instruments very well. But, in recent years there has been great advances in the field of artificial intelligence. Artificial intelligence is made possible by various computer programming techniques, which allow to mimic behaviors considered intelligent. These systems analyze huge amounts of data, which represent real-world information.



Artificial intelligence has made many advances. For some years now, Facebook has been analyzing photographs and trying to predict the presence of people in these photographs; this is known as *tagging* people. Also, we have the autonomous driving systems of some cars, which can get to completely control the vehicle, or at least stop it in case of going towards an obstacle. And we cannot leave out digital assistants (like Alexa, Siri, Cortana, or Google Assistant), which for many is already part of everyday life.

Well, all the systems mentioned can improve as they are in operation, others are updated manually with more and better collected data. Something they have in common is that they work with a sufficient amount of information collected from many places.

One technique that matters a lot in this study is the so-called *Deep learning*, which is a *Machine Learning* technique, is basically to provide a set of data, which consists of two parts: the input data and what is expected to be obtained from it. You could understand how to learn based on examples, but it's more complicated. A classic example: you want a program to be able to identify where a cat is in an image. Initially it relies on analyzing the dark and light areas of the image, then looks for simple shapes (such as lines), then moves to the third level, where it looks for more complex shapes (such as ovals). This process continues until the model manages to identify cats. This example relates to being able to identify printed characters and represent them as computer digital characters.

At this point, the relationship between music and artificial intelligence is beginning to be clear. Human beings can somehow focus our attention on a conversation among several others, this is isolating. The same applies to music: we can concentrate on an instrument, although this is not always easy. This is where *Deep Learning* is useful: use a model that isolates certain tracks from music

As mentioned earlier, Deep Learning consists of having a dataset, where the input data and its expected output are found. In this case, it would have a certain amount of songs with their respective tracks (bass, drums, vocals and others). Based on that

data, the neural network is trained and can isolate tracks from almost any song (not just those included in the dataset).

The basic principle of this project is an Android mobile app that makes use of this technology for this purpose (isolate the mentioned tracks). But, because this process is expensive in terms of processing, the device is generally not able to run it effectively (whether due to lack of memory or processing capacity). To solve this problem, the client-server model is used, thus providing the application with extra compute capacity (server).

Here, the application user must have the audio file of the music whose tracks they want to isolate. With this file, the user must choose the file in the application and it is sent to the server to process; the user only has to wait, and the phone does not use their CPU for this purpose. When the process has completed, the user is notified, and the file is now available from the same application.

The following pages of this document explain in more detail how the entire system works, as well as the software implementation of the *Deep Learning* method. It explains in detail the system architecture, the server that is used to process the songs and how the application works and how it interacts with the other parts.

This project would not be possible without the **Demucs** model, as well as its implementation.

Demucs Study Paper:

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

Code repository with Demucs implementation code:

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

## Background

### Cocktail effect

Cherry was the first to notice the cocktail effect (Cherry, 1953): how the brain can isolate a conversation from a lot of people talking in a room. Bregman then tried to understand how the brain can analyze a complex auditory signal and separate it into higher-level parts (Bregman, 1990).

### Mixing and music mastering

When the music is produced, it is recorded in parts, and these parts are mixed to form the song. It's like when a car is produced: first its parts are manufactured, and then assembled.

Mastering is polishing the mixture. Equalization and limitation techniques are generally used. It's to make the mixture sound as good as possible. Mastering helps us better distinguish the instruments of the song, and therefore, helps the neural network in the separation process.

### Separating music sources in the waveform domain

Unlike the cocktail effect, here is more of an origin of interest, characterized by different timbres and tones.

In the SiSec Mus assessment for music separation (St.e. et al., 2018), the tracks or stems are grouped into 4 categories: drums, bass, others, and vowels. The goal is to generate those 4 tracks from a source (called a mixture).

Demucs is a Deep Learning model, inspired by Conv-Tasnet. Demucs operates directly at the raw input wave level and generates a waveform for each source. It has been proven that operating with the waveform instead of the spectrogram is significantly better. Also, human evaluations were made to Demucs and resulted in better perception.

## HTTP server

An HTTP server, or WEB server, is a program that processes data and sends a response to the person sending a request. Both the request and the response have a similar structure. Works under the TCP/IP protocol. The server listens over a TCP communication port (usually 80). There are many of these servers, and they allow the availability of all websites.

## Process queue

It is a linked list, which contains references to the processes to be executed, one by one, based on different criteria (it may be priority, complexity, or the order in which they arrive). It is especially useful with processes that require a large processing capacity and memory, where if done in parallel, memory could run out and cause the system to drop. Variants can be implemented where more than one process is running at a time, it all depends on the situation.

## Objectives

Developing the **Isolaatti** application aims to:

- Allow users to use Demucs easily, from the convenience of a graphical interface on their Android devices.
- Provide an easy-to-understand interface, following Android's graphical interface design standards. This to facilitate the user's understanding of the application.
- Provide the possibility to manage a simple mixer in the application, allowing the user to mute or activate the tracks in real time.
- Be of use to everyone, especially musicians.
- Allow the use of the application on different devices that are owned, through user accounts that save preferences and projects (songs with separate tracks).
- Allow projects to be shared to other users, even if they are not registered.
- Allow the user to easily equalize each track.
- Allow the future expansion of the service. This means that it is possible to develop a WEB client, a client for iOS, etc., thanks to the development of a WEB API.

In summary, the application aims to support musicians, so that they find it less difficult to learn a song they want to perform with their instrument.

## Goals

When developing Isolaatti I have how long-term goals:

- Get the Android app to a stage where the operation is perfect.
- Make everything behind the app more robust.
- Ensure service availability all the time.
- Improve service. This includes decreasing processing time and handling errors that may come out on the server, so that it does not affect users' files.
- More servers: to be able to process in parallel, which means avoiding long waits for users.
- Get the app to more users.
- Monetize the app with ads.
- Integrate Google Drive into the app.
- Allow users to communicate with each other. This limited to sharing songs with friends, but without relying on another messaging service or social network.
- Add the ability to complete song metadata with information from a metadata API.
- Allow you to locate the lyrics of the songs, directly from the application.

the purpose of Isolaatti is to help musicians, but also people who are curious or interested in music. But, for ease of use, anyone can use it without further complication.

## Sustainment of the realization of the project

### Decreased expenses for musicians

Many times, musical groups haven't to buy isolated tracks, and such tracks are not cheap. With Isolaatti, you can try to isolate the tracks, and you usually get good results in the following:

- Isolate drums
- Isolate bass
- Isolate voices
- Combinations such as:
  - Remove voices to generate instrumental tracks
  - Remove the bass
  - Remove the drums

The above can be done directly from the same application, previewing the mix.

### Supporting technological knowledge

Society is increasingly aware of what the advancement of artificial intelligence allows. Isolaatti informs the user of the imperfections of the system, but also how it works broadly. This helps the user increase their knowledge of what AI and distributed and client-server-based systems can do.

### Experience planning a software project

Planning a software project, whatever it is, is quite a process. You cannot go straight to the code, but first you must conceive all parts of the system.

## Development experience

This is one of the first major projects I developed. Logically, programming, configuring, and implementing all parts has put me to study different technologies, which if I had not done a project like this would not have studied (at least for now). From programming for Android, to setting up a server on Linux.

## Experience of providing a service

It's definitely not the same to make a local app reach many users and have them have a pleasant experience, which make an application that relies on multiple servers stand and a user-friendly experience.

Indeed, providing a service like this requires money. Monetizing the app is a good way to earn revenue to keep everything running. Of course, good programming is very important to make the most of computational resources.

## Summary

As a social impact I classify both the support for the reduction of expenses for musicians in the purchase of instrumental tracks, but also the impact for other users (non-musicians) as well as the impact that I have experienced as a software developer.

Developing this project is a step towards my future as a professional software developer.



## Development methodology

One of the hardest things, which you need to do to make the service available to a larger number of people, is to use multiple machines running the process server.

A single machine with the current hardware (dual-core processor and 8GB of RAM), can process one song at a time, this limited by memory. According with measurements taken:

$$t(x) = 1.3x$$

Where  $t(x)$  the machine takes time and is the duration of the song in  $x$  seconds.

To the above expression it is necessary to add the time required to decode the source audio (this process is done before the separation begins) and the time required to encode the generated tracks to mp3 (this is done after the .wav)).

Decoding time is on average

$$td(x) = 0.216x$$

The encoding time to mp3 is negligible

So, let's say a song lasts on average 3 minutes. Let's calculate how many songs could be processed:

First, it calculates the time it would take for a 3-minute song to process

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

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

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

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

The song would take approximately 272.88s or 4,548 minutes or

**4:32 in clock notation**

Based on the above:

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

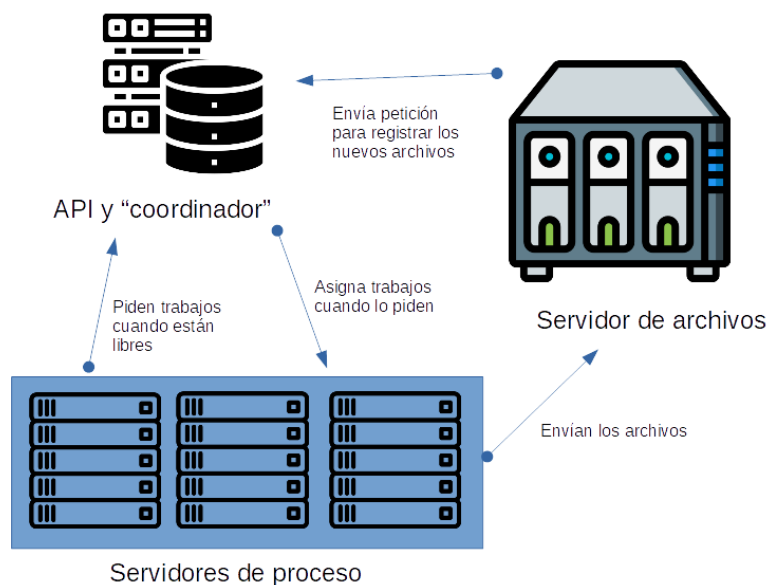
On average, up to **13 3-minute songs per hour** can be processed.

One of Isolaatti's (and mine) goals is to be able to support more users than a few friends. Based on the above calculations, it is easy to realize how many equal machines would be needed to support x number of people.

The system currently has a server that receives all jobs. This could get better if you have more machines. To make things efficient, machines should no longer receive jobs, but ask for them when they are unoccupied; the central server must coordinate them.

Suppose you have 10 machines; this would allow you to process up to 130 songs for 3 minutes each.

Another important aspect is storage. Right now, everything is left on the single process server, so this machine also serves as a file server. This is not a good idea if you have multiple process machines, so it's best to have a single file storage dedicated to it.



*Improved system (not current)*

## Gantt Chart

[illegible]

# Deliverable Products

## Phases

### Programming the process server script

This is programmed in Python. The Flask web micro framework is used to receive the files and glue the songs. You have a process queue, which calls Demucs song by song (one at a time)).

You have a thread that checks for a new item in the queue, and if there is, it sends it to process. This part is very important for the server to work well.

The number of threads demucs can take is limited to 1, as the machine has only two parallels, and if the two are used, the server could not respond well to requests, when clients send songs to process, resulting in very high response times or 500 or similar errors.

### API programming

This is consumed by the application (client). Allows the client to create records in the database. In addition, it takes care of sending notifications to customers. The technical manual explains how to consume this API, specifying HTTP verbs, routes, parameters, and what each thing is for.

### Database configuration

Entity Framerowk Core (running on the same machine as the API) creates the tables and relationships in the database. This avoids the need to hardcode SQL queries, which can be unsafe.

### API de la API deployment

A free Azure App Service is used. Currently here is the database (SQLite for testing).

### Deploying virtual machines with the process server script

A Standard D2s\_v3 plan (2 Cores, 8 GiB of memory) is being rented from Azure. This is running Ubuntu 18.04 64-bit. It has 30 GB of storage.

All the configuration of this machine has been done using SSH. The server script was configured as a service, so that it is running whenever the machine is powered on.

An Apache web server has been installed, which is a file server.

### Design and programming of the Android app

This stage is the one that takes the most time; you need to handle a lot of things related to Android. The basic function is to allow the user to upload songs to the server to process them and get all four tracks. The user manual specifies how to use it, and technical manual explains how it works internally.

## Application architecture

Look at the diagram. The current architecture of the system is composed of three fundamental parts. In the diagram you can see the interactions of those parts.



## Technologies used

### Development machine

It has used a conventional PC with Ubuntu MATE 18.04 installed. A fourth generation Intel Pentium processor with 8 GB of memory installed. It is a PC assembled by pieces.

### Android Studio

This is the ideal IDE to develop Android apps. It is used on Linux for better fluidity, due to the slowness with which it runs on Windows on tight hardware.

### Version control

Git and GitHub are used to version the source code of the different components (Android app, server, and API).

## Cloud service

Microsoft Azure is used for all hosting services. This includes the free App Service with a MySQL instance (which is not used for now), and a virtual machine running Linux.

## Google email account

A Gmail account is used to send account verification emails as well as alerts to users. This account is a personal account, created for this purpose. It's not needed a dedicated email account in testing and development.

## Google Firebase Cloud Messaging

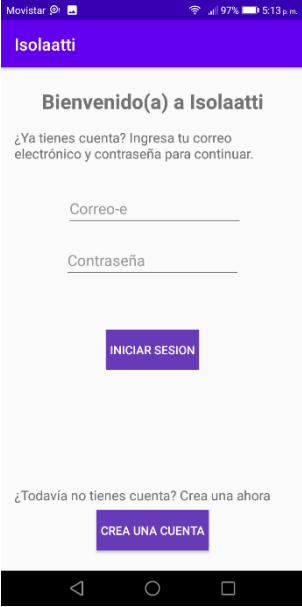
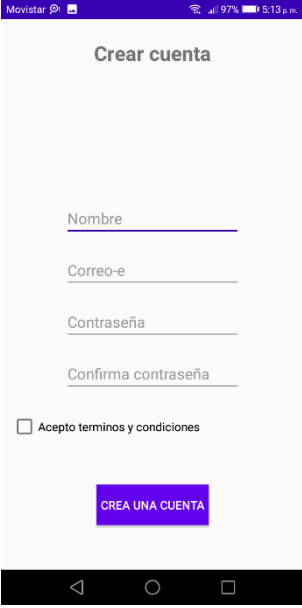
This service is used to send notifications to users. Just have your device's Google token. It should be noted that the user's Android device must have Google services installed in order to use this.

## Languages used

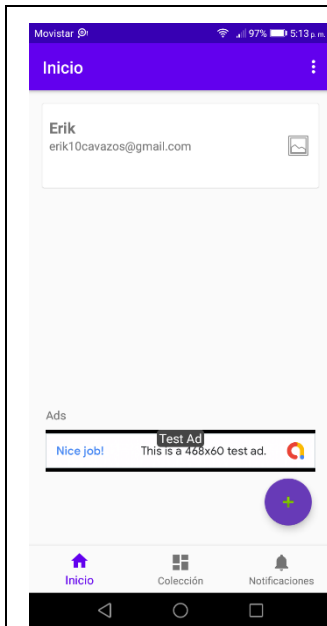
The following programming languages are used:

1. Kotlin: for Android app development
2. C#: for API development. Build under .NET Core 3.1
3. Python: for the process server. Python 3.7 is used
4. HTML5 (HTML, CSS, and JavaScript)– To a lesser extent, for the web development required for compartment links to work (to send to others).

## Screen design

Screen	Brief description
 The login screen features a purple header with the 'Isolaatti' logo. Below the header, the text 'Bienvenido(a) a Isolaatti' is displayed. A prompt asks if the user has an account and provides instructions to enter email and password. There are two input fields labeled 'Correo-e' and 'Contraseña'. A purple button labeled 'INICIAR SESION' is positioned below the fields. At the bottom, a link '¿Todavía no tienes cuenta? Crea una ahora' is shown next to a purple button labeled 'CREA UNA CUENTA'. The screen is framed by a black Android navigation bar at the bottom.	<p><b>Login screen</b></p> <p>Here the user must enter their credentials. If you do not have an account, the user must tap the "CREATE AN ACCOUNT" button.</p>
 The create account screen has a purple header with the title 'Crear cuenta'. It contains four input fields: 'Nombre', 'Correo-e', 'Contraseña', and 'Confirma contraseña'. Below these fields is a checkbox labeled 'Acepto terminos y condiciones'. A purple button labeled 'CREA UNA CUENTA' is at the bottom. The screen is framed by a black Android navigation bar at the bottom.	<p><b>Create account screen</b></p> <p>Here the user must enter the requested data to create an account. Afterwards, an email is sent to the address that was provided, this in order to verify it.</p> <p>If the user does not verify the account, this account will not work.</p>

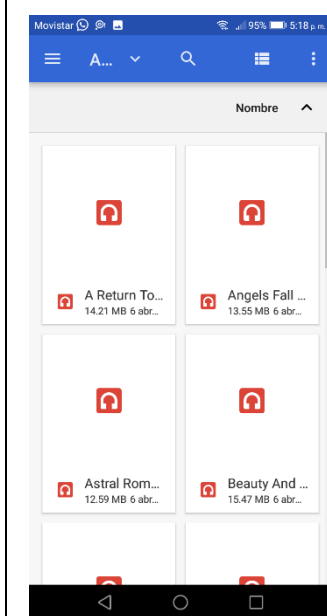




## Home screen

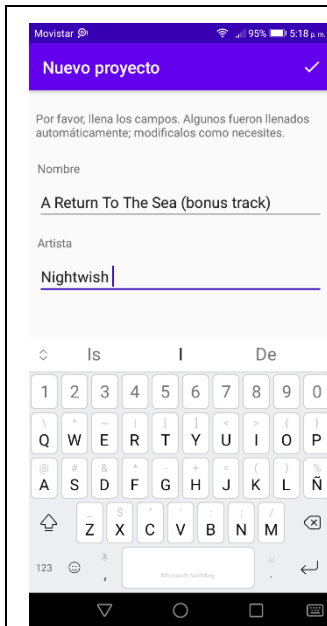
This is where the user arrives when they launch the app from the launcher. Only your profile information and a button to go to the file upload screen are displayed.

NOTE: Tapping the box where the profile information comes from accesses the profile screen, from where you can change the password and other things.



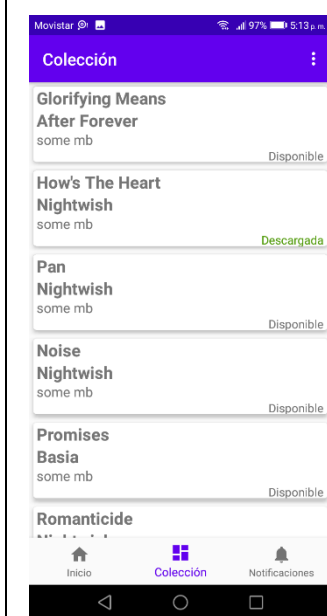
## File upload screen

This screen is provided by the phone, so the design may vary. This is where the user chooses which song they want to upload to make a separation project.



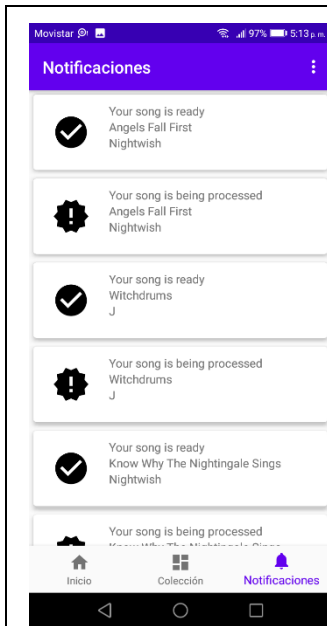
## New project information screen

This screen is presented after the user chooses their file. Here the user must make sure that the song data is found as they want it to appear in their collection.



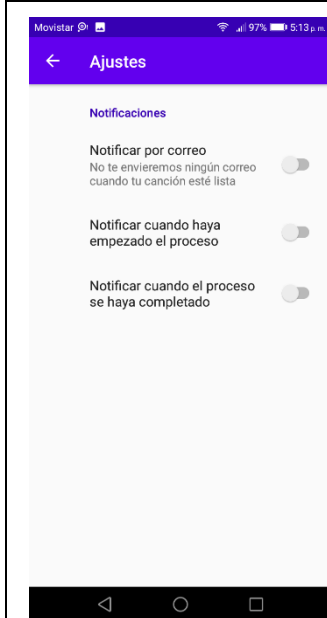
## Collection screen

Here are the songs the user has in their collection. From here you can delete, edit metadata, and open them.



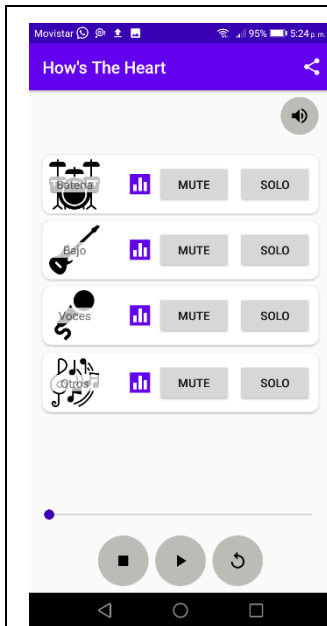
## Collection screen

Here is the history of notifications that have reached the user, as well as those that were sent, but not received at the time.



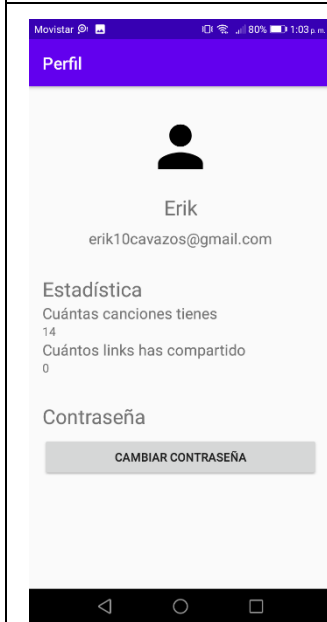
## Settings screen

The application settings are concentrated here. Now, you can only control notifications.



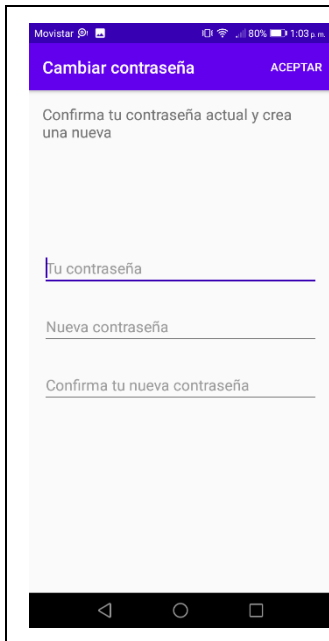
## Track player screen

This screen is the most important of the application. From here, the user can generate the combinations they want with the tracks. In addition, it has a simple equalizer per track.



## Profile screen

Here the user can see their name, mail and a count of songs and links. Also, from here you can change the password.



## Password change screen

This screen is used to change the password. If the process is successful, you will be automatically logged out and the user will be prompted to log back in with their new password.

## Implementation

All servers are listening on the Internet. The Azure cloud services and Google Firebase have been used. The app isn't really available yet to every user; it's found in its own repository. In this repository only APKs are uploaded APKs.

## Modules

The application consists of the following modules:

1. General module
  - a. Start new project
    - i. File selection
    - ii. Data entry
  - b. Change account information
    - i. Changing your password
    - ii. Username change
2. Song module
  - a. Collection
    - i. Delete song
    - ii. Modify song information
  - b. Track player
    - i. Equalizer
    - ii. Share
3. Notification module
4. Settings module
5. Account module
  - a. Account creation
  - b. Log

## Libraries

The following libraries are used in the application:

- Volley– Library for making HTTP requests to servers. <https://developer.android.com/training/volley>
- Android Upload Service: to upload files by POST. <https://github.com/gotev/android-upload-service>

On the API side the following libraries and SDKs are used

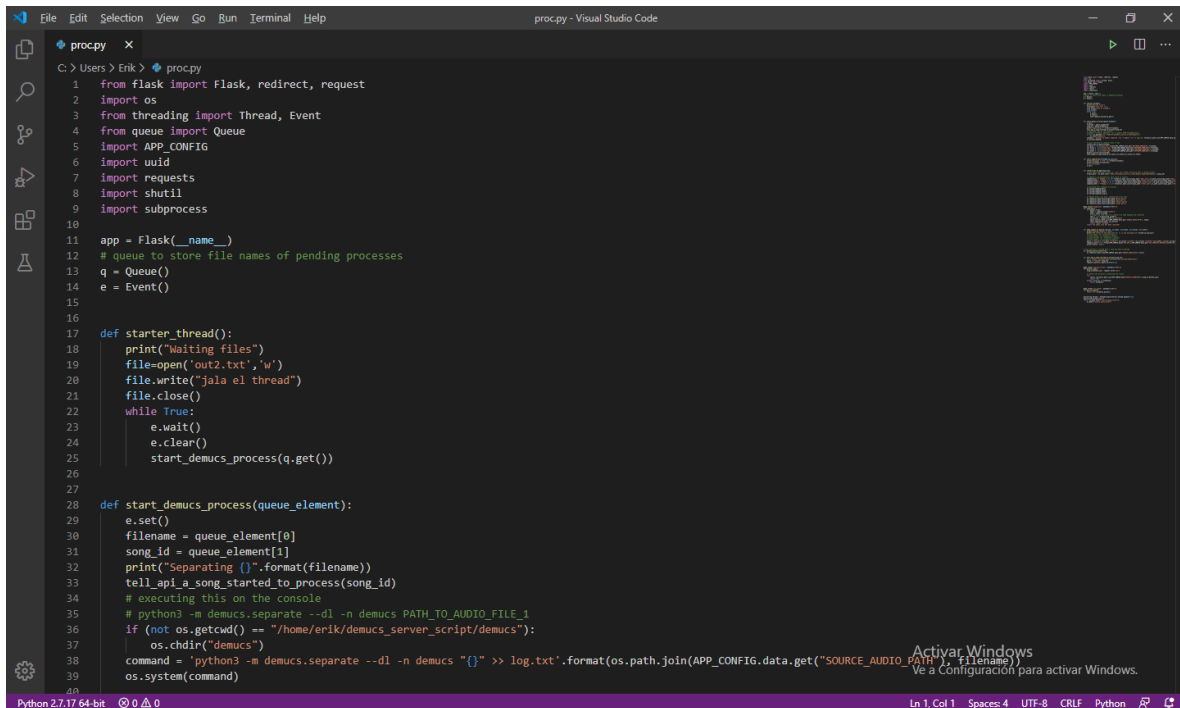
- Firebase Admin SDK for C-: To be able to send notifications to devices from the server. <https://firebase.google.com/docs/admin/setup?hl=es#c>

The following micro-framework is used on the processing server:

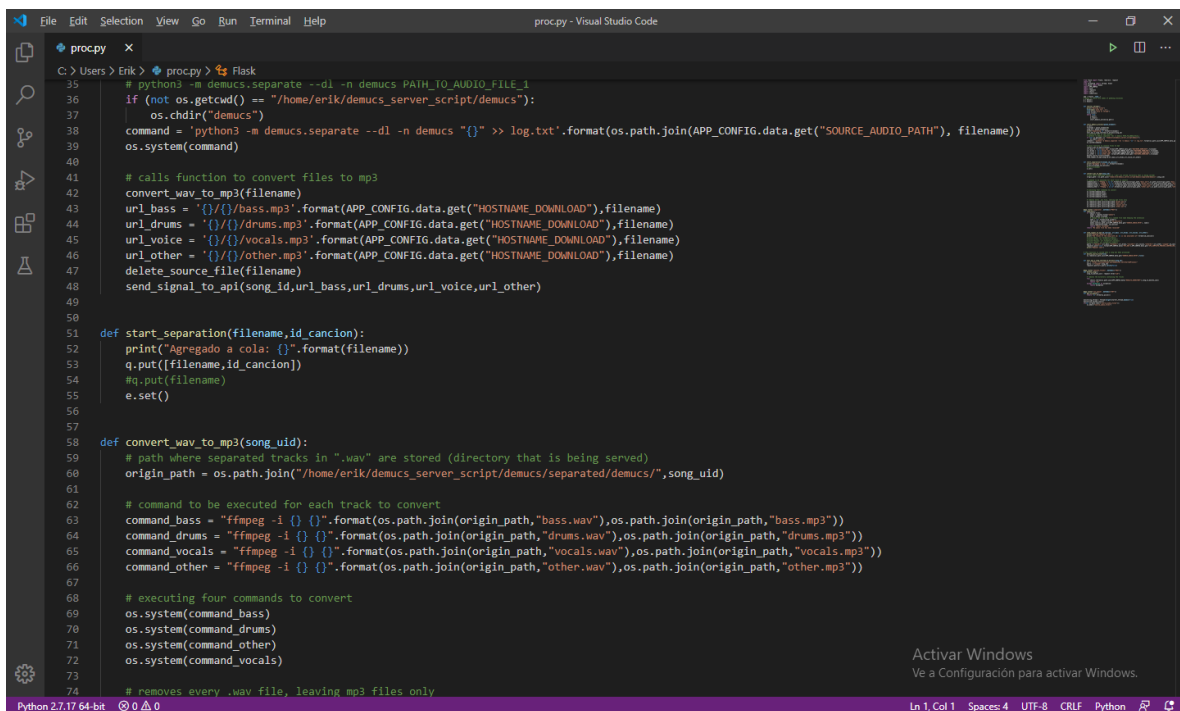
- Flask– To handle HTTP requests. <https://flask.palletsprojects.com/en/1.1.x/>

# Scripts

Next, I provide a sample of the Python script code for the process server.



```
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)
```



```
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_vocals = '{}({})/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_vocals, 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
```

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

C:\Users\Erik> proc.py > send_signal_to_api
os.system(command_vocals)

# removes every .wav file, leaving mp3 files only
os.remove(os.path.join(origin_path, "drums.wav"))
os.remove(os.path.join(origin_path, "bass.wav"))
os.remove(os.path.join(origin_path, "vocals.wav"))
os.remove(os.path.join(origin_path, "other.wav"))

@app.route("/separate", methods=["POST"])
def separate():
    if request.files:
        audio = request.files["audio"]
        name = audio.filename
        # change the file name to a random file name keeping the extension
        name = "{}".format(uuid.uuid4())
        id_cancion = request.form["id_cancion"]
        audio.save(os.path.join(APP_CONFIG.data.get("SOURCE_AUDIO_PATH"), name))
        start_separation(name, id_cancion)
        return "Audio recibido"
    return "No audio file has been received"

def send_signal_to_api(id_cancion, url_bass, url_drums, url_vocals, url_other):
    print("Sending request to API")
    print("The tracks of the song with id: {} is now available at".format(id_cancion))
    # print("Bass: {}".format(url_bass))
    # print("Drums: {}".format(url_drums))
    # print("Voices: {}".format(url_vocals))
    # print("Other: {}".format(url_other))
    query = {"bassUrl":url_bass,"drumsUrl":url_drums,"voiceUrl":url_vocals,"otherUrl":url_other,"songId":id_cancion}
    requests.post("{}{}".format(APP_CONFIG.data["API_URL"],APP_CONFIG.data.get("API_COMPLETE_PROCESS_CONTROLLER_NAME")),query, verify=False)
    print("Request sent")

# this function is called when a song has been processed
def delete_source_file(file):
    os.remove(os.path.join(APP_CONFIG.data.get("SOURCE_AUDIO_PATH"),file))

def tell_api_a_song_started_to_process(song_id):
```

Demo code available for viewing at:

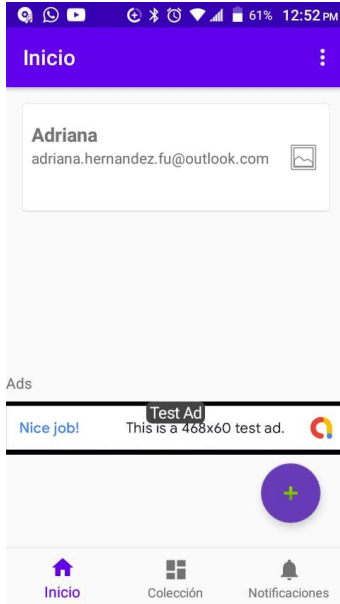
<https://drive.google.com/file/d/1ZZUN1HITtcmTu2jNRidTCRNZuawRcADX/view?usp=sharing>



## Customer testing

The Android app was tested on the following phones:

### ZTE Blade L7A



**Android version:** 7.0 Nougat

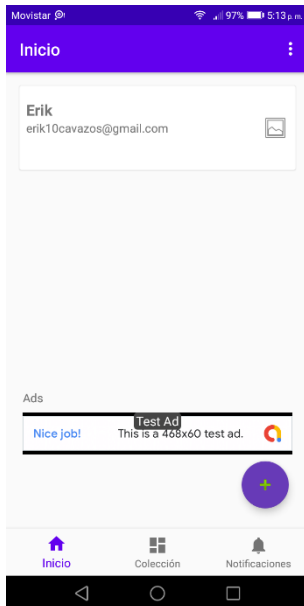
**RAM:** 1 GB

**Storage:** 8GB

**Display:** 5 inches, 480 x 854

**Processor:** Spreadtrum SC7731 Quad Core @ 1.2GHz

### Huawei Honor 7S



**Android version:** 8.1.0 Oreo

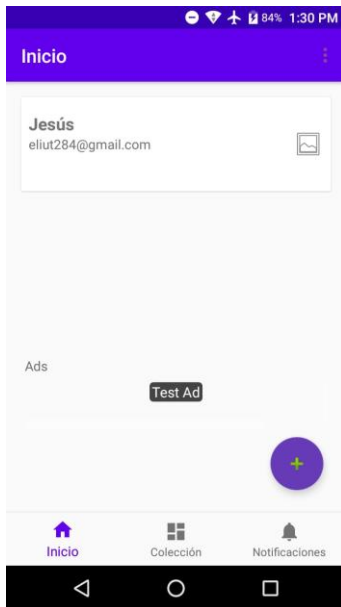
**RAM:** 2GB

**Storage:** 16GB

**Screen:** 5.45 inches, 720 x 1440

**Processor:** Mediatek MT6739 Quad Core @ 1.5GHz

# M4 SS4458



**Android version:** 6.0.1 Marshmallow

**RAM:** 2GB

**Storage:**16GB

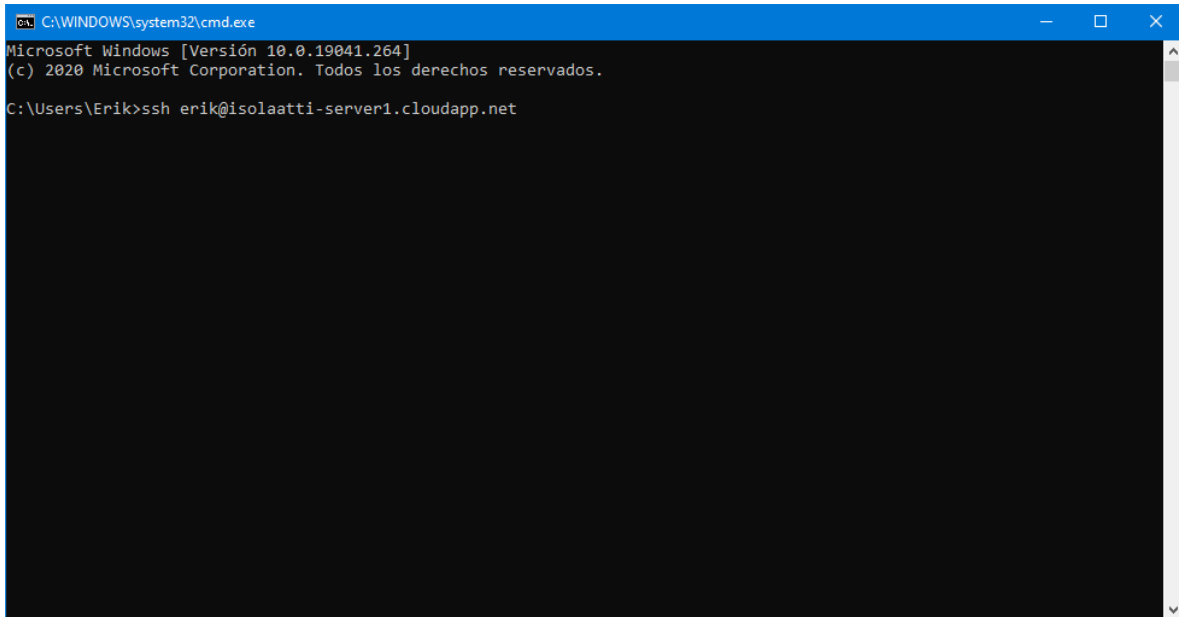
**Display:** 5.5 inches, HD (720 x 1280)

**Processor:** Qualcomm Snapdragon 212 Quad Core @ 1.3GHz

## Testing on the server

The song processing server (the one running in Python) is currently listening on: `isolaatti-server1.cloudapp.net`. The technical manual specifies how to use the paths of this web server.

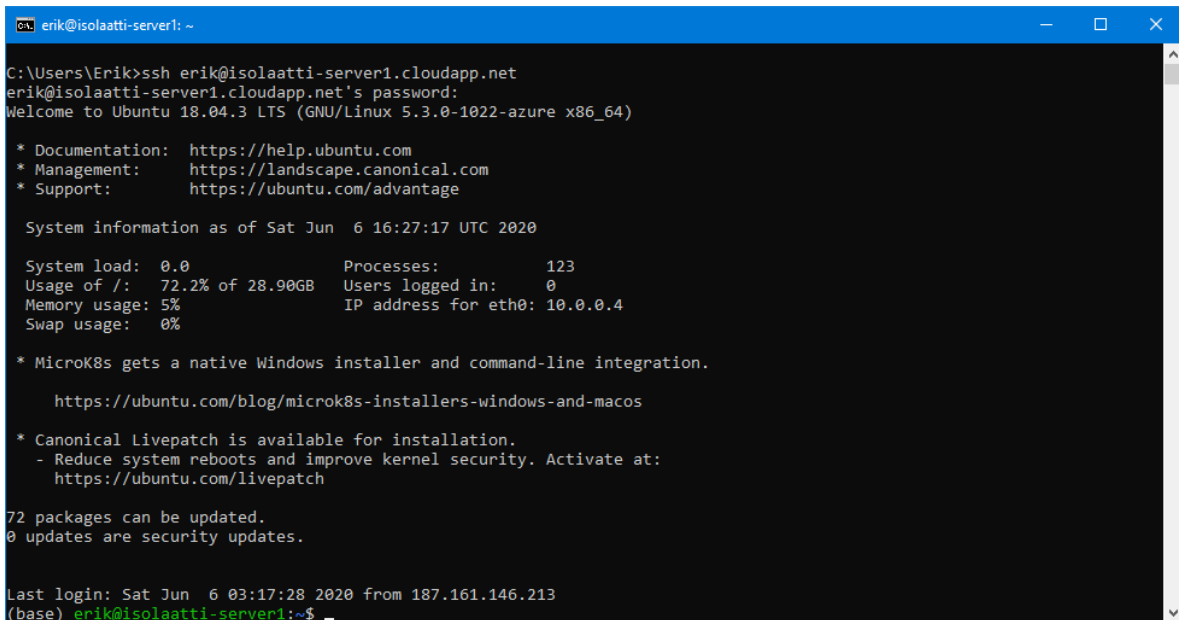
This is how to log in to the server by SSH



```
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
```

Here is the session



```
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:~$
```

Using the *htop* program to view the processes. The remarked process is one of the server threads.

```
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 0:00.00 /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 nobody 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
```

The process with PID 64680 screenshot is demucs running. Note that you use 100% of one of the cores and a large amount of available memory.

```
erik@isolaatti-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 100.0 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-ext  
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-ext  
1297 root 20 0 229M 2040 1480 S 0.0 0.0 0:00.13 /usr/bin/lxcfs /var/lib/lxcfs/  
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/linux-tools/5.3.0-1022-azure/hv_kv_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-ext  
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/ssh -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
```

## Referencies

(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>

## Abstract

Isolaatti is an Android application that uses the demucs neural network (from Facebook research) to isolate the drums, bass, voices, and others of a song. This neural network is behind a web server, which is responsible for receiving the music files and sends them to process to the neural network one by one.

In the application, you have user accounts, song collection and most importantly, a track player, which allows you to listen to the four tracks at the same time and mute and equalize the ones you want, generating the desired mix in real time.

The application sends the music files to the process server using HTTP, using the POST method. The tracks obtained in the neural network are made available to the user for download in the application. The application downloads them and allows the user to listen to them from there.