# Bulbul

CS 492

## Final Report

**Group Members:**

Enes Akdoğan – 21200514
Mesut Gürlek – 21201157
Ömer Akgül – 21301854
Fatih Taş – 21300836
Ali Burak Erdoğan – 21301492

**Supervisor:** Mustafa Özdal
**Jury Members:** Gökberk Cinbis, Uğur Doğrusöz

bulbulproject.com
**04.05.2017**

# 1. Introduction

We live in a world where music has almost become an essential part of most people’s lives. Whether you are sitting on a bus or crunching numbers for your next math homework chances are that you are listening to music. Of course everyone has their own taste in music and tries to select his/her music depending on this. However, this can be frustrating at times. Our aim is to develop a music recommender system integrated in a mobile music player application which would ease this decision dilemma. If we take a closer look, it is obvious that music is related with a considerable amount of metadata such as musician information, genre type, release date, music length etc. It can be possible to create a recommender system with just the metadata where a user would only get recommendations from the types of music he/she previously listened to. This might seem like a logical model at first but after a while the system either becomes too repetitive or too unpredictable which most possibly renders the recommendations useless.

This report aims to give an overview of the finalized bulbul system.

# 2. Design of the System

## 2.1 General WorkFlow of the System

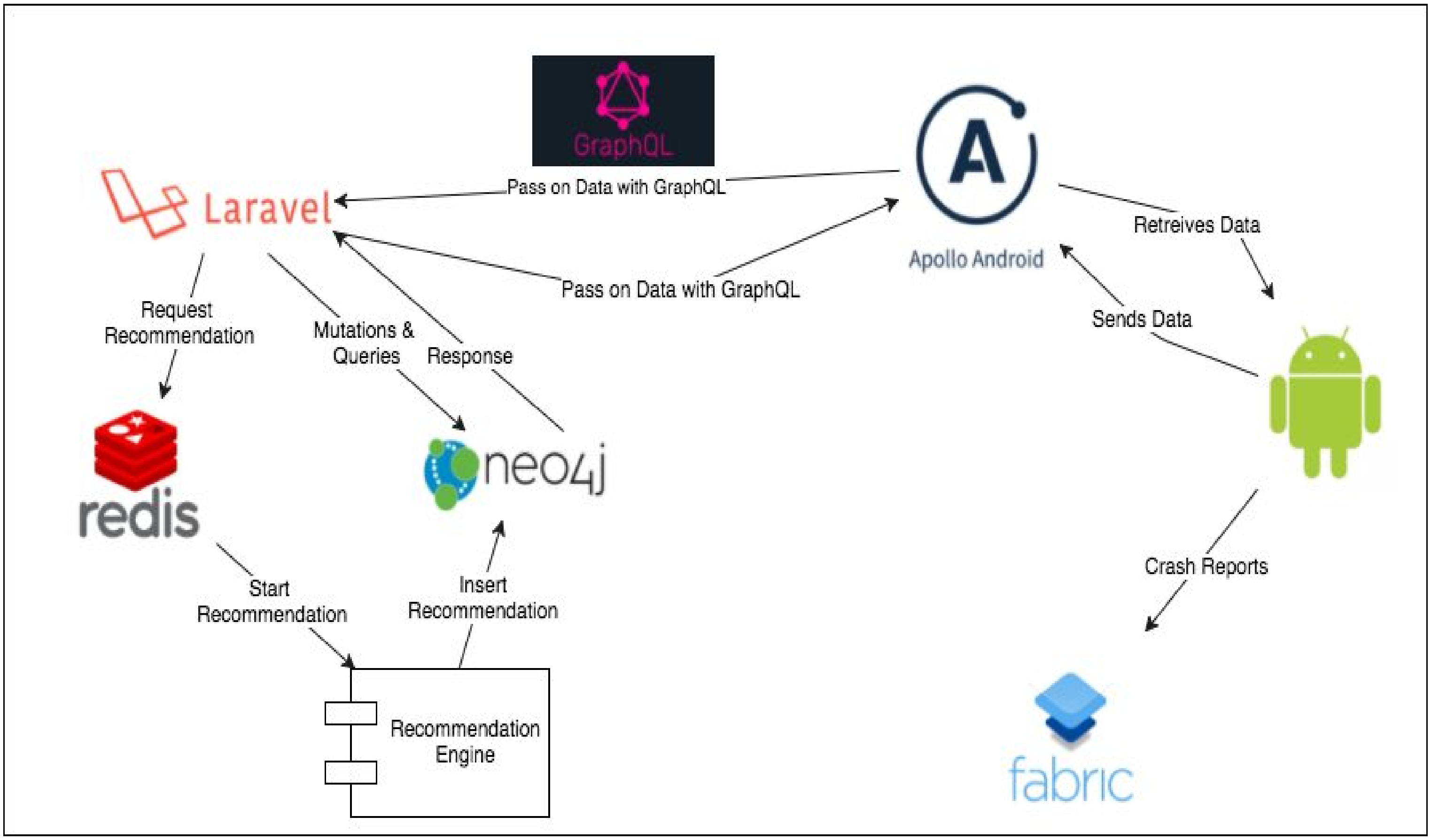


Figure 1: General flow of data in the bulbul system.

In figure 1 we can see the general workflow of the Bulbul system. We can describe all the components and how they are connected as follows.

The android logo represents the Bulbul android application. This is where the user interacts with the system by listening songs, rating them, and managing them via playlists. The application provides a simple to use interface to see the user's songs, albums, artists, playlists, and recommended songs. We manage all crash reports in the system via fabric [1]. Fabric basically logs all crash reports to their online system which we can access on demand. This makes it easy to fix common bugs without actually having to replicate problematic situations. The data fetched from the database and sent back are done so using the Android library Apollo [2]. Apollo allows us to receive and modify data as Java objects in Android, this is very convenient when coding an application since we used Java as our main programming language. Apollo translates incoming GraphQl (more about GraphQl will be discussed later.) queries to Java objects and does the opposite as well.

The data and running recommendation algorithms are kept on two virtual servers both hosted by Digital Ocean [3]. The first server is to keep the the graph database online while the second one is to serve the recommendation requests received from users. Both servers are currently work at full capacity and are easily expandable thanks to the easy memory and disc space expanding options Digital Ocean provides.

For the database we use a graph database known as Neo4j [4]. This allows us to map all of the relations between users, songs, albums, artists, playlists etc. The query language for Neo4j is known as Cypher. It is an easy to write and understand language however for our purposes we decided to use the query language GraphQl [5]. GraphQl basically allows us to use a new API type which makes writing queries and receiving responses much easier compared to a traditional Rest or Soap API. In order to connect GraphQl queries to Neo4j -and thus implementing the API- we used the PHP framework Laravel [6]. Laravel connect structures in the Neo4j database to the GraphQl system so that Cypher queries can be translated forward and back. Laravel framework and the Apollo library are similar in the sense that both exist to translate queries for their platforms. In Short, Neo4j is our graph database and the transactions between the database are provided through Laravel, GraphQl and Apollo.

Recommendation requests from the Android application are also managed through the existing Apollo, GraphQl, Laravel system. Basically when a recommendation request is detected by Laravel the request is passed on to Redis [7]. Redis can be thought of a queue of requests and when the queue is not empty a connected script is called to serve the recommendation request. After the recommendation calculations are made the mentioned script pushes the recommendations to Neo4j. This way the recommendations are represented as relationships betweens users and songs and a query from the Android side can easily access it. Redis allows us to attach multiple sessions to it thus, with sufficient hardware the recommendation system can be served by multiple workers.

The overall system architecture is described as follows.

Bulbul has a multi-layer architecture. Layers are organized and implemented independent from each other. Multi-layer architecture provides scalable, maintainable and extensible system. It consists of 4 different layers:

- Presentation Layer: This layer contains the user interface and corresponds to Bulbul's Android application client. Users interact with the presentation layer. They can view musics, recommendation and discoveries from this layer.
- Application Layer: It is an intermediate layer between Presentation layer and Data Storage layer which works as an API. We are using recently popular *GraphQL* technology and *Laravel* framework to establish an API. It collects data from user (like ratings given to tracks, recently followed musicians etc.) to store in database for doing Data Processing later and sending generated information such as recommended musics, auto-generated playlists etc. to Android client.
- Data Storage Layer: This layer stores the data that comes from application layer. Since the nature of our data has many relations between entities such as user-to-user followings, user-to-artist, user-to-genre subscriptions etc. we use *Neo4j graph database* to easily process our dataset. That way, we become able to easily detect from our dataset:
    - ❏ social networks,
    - ❏ people being similar in terms of music taste,
    - ❏ or similar artists regarding music genre.
- Data Processing Layer: This layer processes data and make recommendation computations. Collaborative filtering and content based recommendation algorithms run on our collected data in this layer. This layer is built upon our

cloud server. We establish the data transfer between data storage layer and data processing layer using *Redis* which provides messaging and caching utilities.

# 3. Core Algorithms

## 3.1. Dataset and the Fetching Process

We constructed whole dataset by our efforts. LastFM API provides lots of information about their users and their music history. Also track, album and artist metadatas are easily accessible from the API. Out data collection methodology is as follows: We manually detected communities and fetched user informations. Based on user metadata, we generated unique track list that contains all tracks that users listened. Then  based on these track lists we generated unique albums list and artist lists. After generating these lists we used 6 fetcher cloud machine to fetch all information from LastFM API. After fetching metadata we needed to stream these tracks. Hence, we used Spotify API to get streaming urls. We first used search functionality of the API to match our tracks  with spotify tracks by using track name, album name and artist name. We got 80% match and fetched all track streaming link and song features from spotify. This process took **1.5 month** for us. Finally we got the following number of data:

- ~15.000 Users
- ~900.000 Tracks
- ~240.000 Albums
- ~90.000 Artists
- ~35 million Relations between Nodes (Graph Database Neo4j)

Figure 2: LastFM and Spotify Logos

## 3.2. Real World Dataset Challenges

Since we are constructing whole dataset from scratch, there were many inconsistencies for the algorithm. We tried to analyse data during database insertion and algorithm adaptation. We used Graph database Nep4j, so each table becomes a node and relations between nodes should be consistent if we are trying to get meaningful recommendations. Therefore, during insertion we get rid of some noisy data by the help of our inspection and scripts. Also, during the development of the algorithm, we got several exceptions because of the inconsistencies in the data and handled them in the code.

## 3.3. Recommendation Algorithm

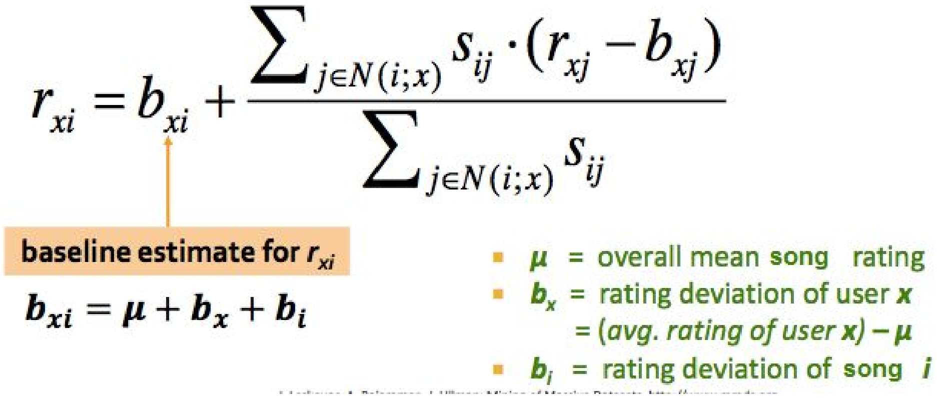
The recommendation feature is the flagship feature of the Bulbul system. In order to avoid a cold start problem with the algorithm we gathered music to user relation data from last.fm apis. The current data we have, describes the most listened songs of each user as well as the most liked songs of each user. This allows us to generate a rating for each song for each user. The recommendation algorithms described below use this existing rating data to make recommendations. For the content based recommendation system we gathered metadata for each song using the Last.fm api as well as the Spotify api.

### 3.3.1. Item to Item Collaborative Filtering

We have implemented Item to Item Collaborative Filtering which is the main recommendation algorithm. Algorithm works as follows: First, we should obtain rated song list of a user. Then we create a utility matrix which composes of users and their ratings to all songs in the database. We find similar songs that user never listened before to user’s most liked songs. Then we estimate user’s possible ratings for that similar songs, and return highest rated musics as recommendations.

### 3.3.1.1. Rating Estimation

We implemented Item to Item algorithm in two different ways. In first way, we generated ratings for each song that users listened according to metadata we have. Then we implement described algorithm above and return top rated songs as recommendation. We use following formula to estimate rating that user gives to a song which he did not listen:



$$r_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$

baseline estimate for $r_{xi}$

$$b_{xi} = \mu + b_x + b_i$$

- $\mu$ = overall mean song rating
- $b_x$ = rating deviation of user $x$ = *(avg. rating of user* $x$*)* $- \mu$
- $b_i$ = rating deviation of song $i$

Figure 3: Collaborating Filtering Item-to-Item Formula

-*sij:* similarity of items *i* and *j*

*-k nearest neighbors N(i; x)*

*-rxi: as the weighted average*

### 3.3.1.2. Binary Estimation

In second way, we created utility matrix from binary liked(1) or not liked(0). Hence, we did not estimated the ratings that user could give to songs, but we estimated user can like or not a song that he/she did not listened.

### 3.3.2. Content Based Recommendation

We have several music features which identifies songs' danceability, speechiness, acousticness, instrumentalness etc… We used these music features and create unique vectors for each song. Since we know the songs that user loved, we recommend similar songs with using these feature vectors. We find similar songs and return highest similar ones as recommendation.

### 3.3.3. Neo4j Quick Recommendation

We are storing our data in a graph database called Neo4j. We find similar users to current user making a query and the return top rated songs from similar users as recommendation.

### 3.3.4. Ensemble of Algorithms

For the final recommendation algorithm we use a combination of the previously explained three methods. This allows the system to take advantage of the strong points of each recommendation algorithm.

### 3.3.5. Improvements in the Algorithm

Since we are using a huge dataset, we are also generating huge matrices to run our graph algorithms. These each matrices are around ~200 mb, so we should avoid from loading them in every run of the algorithm. That is the reason why we cached all matrices before running algorithms. We used redis to cache them so we load matrices once and they are always in the memory.

Furthermore, time is very important for us to calculate real time recommendations. Therefore, we analyzed each code snipped in the algorithm in terms of time and optimized them. Now, we are able to give real time recommendations in seconds!

### 3.3.6. Exploration vs Exploitation

We will provide different services which have different trade offs. User can exploit the popular songs to get quick recommendation but not very original or personalized. Conversely, user can explore deep inside the dataset to  get unique and personalised musics using cutting edge recommendation algorithms but it might take some time.

### 3.3.7. Distributed System with Redis

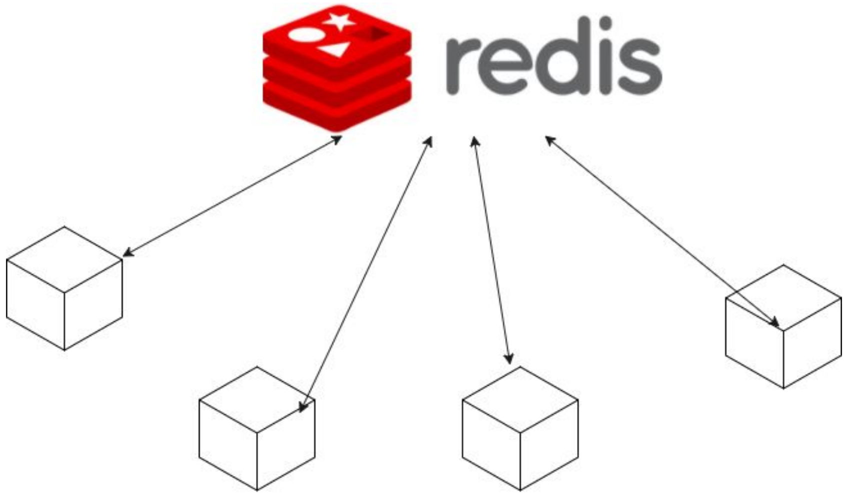


Figure 4: Redis Multi Cloud Machine System Representation

- All recommendation requests(tasks) are kept in Redis Queue
- Multiple worker machines can fetch requests in parallel
- Easily Scalable since it is Distributed

### 3.3.8. Accuracy Testing

We used some accuracy metrics to test our recommendation algorithms such as: SSE(Sum of Squared Error), RMSE(Root Mean Squared Error)

# 4. Tools and Technologies Used

## 4.1. Neo4j

It is a highly scalable, native graph database purpose-built to leverage not only data but also its relationships.

## 4.2. GraphQL

GraphQL is an application that takes input as a string for query languages and returns the data in a specified format.

## 4.3. Laravel

Laravel is open source PHP web development framework. Laravel follows MVC (Model-View-Controller) philosophy.

## 4.4. Redis

Redis is an open source in-memory data structure store and used as a database. It keeps data as key-value pairs. It is used for job queues.

## 4.5. Apollo

Apollo is Android library that generates Java models from standard GraphQL queries. So it gives API service to work with server safely.

## 4.6. Digital Ocean

DigitalOcean is a cloud infrastructure service which is focusing on simplifying web infrastructure.

## 4.7. GitHub

GitHub make easier to coding, testing and deployment within a team. It uses Git as a VCS (Version Control System). Git provides many features which are very useful while working as a team. Team members can work on different parts of the project independently thanks to using Git.

## 4.8. Python

Interpreted programming language. Used in the server side implementation of the recommendation algorithms. This language was also used in order to gather data from the Last.fm and Spotify apis.

## 4.9. Android Studio

Android Studio is an IDE (Integrated development environment) which is used in Android Application Development. It is based on IntelliJ IDEA and freely available.

## 4.10. PhpStorm

PhpStorm is and web development IDE of JetBrains. PhpStorm provides an editor for PHP, HTML and JavaScript etc.

## 4.11. PyCharm

PyCharm is and python development IDE of JetBrains.

## 4.12. Atom

Atom is an open source editor project and it works on macOS, Linux and Windows.

## 4.13. Sublime Text 3

Sublime Text is a cross-platform source code editor.

## 4.14. Trello

Trello is one of the most used project management tools. It works on many devices perfectly synchronously. Project manager can assign subtask of project to other engineer in that project, and system push notifications to the engineer for his/her new task. All members of project can see progress of the project and which tasks have done and which of them still developing and see future plans for that project. Trello provides boards to control project. Projects are consisted of boards, boards are subtasks of that projects. Boards are also composed of cards. Cards can be assigned to members and his/her progress can be controlled by his/her supervisor. Trello provides checklists, to-do lists, labels due dates and many feature for project members to make project management easier.

## 4.15. Slack

Slack is a real time messenger application for teams. Slack is easy to learn and provides many features. Slack can works with many other tools effectively. We integrated GitHub, Trello and our scripts to Slack and teammates follow process from these channels easily.

## 4.16. Fabric

Fabric provides a Crashlytics tool to monitor our app and logging the crash report from all user. Thus, we are able to view all crash reports, from all users and fix these problems quickly.

# 5. Engineering Solutions and Contemporary Issues

Recommender systems became more popular in recent years as the number of items over internet increases beyond a user can reach with other systems like search engines. Using a recommender system in music industry allowed us to help users discover songs that match exactly with their personal preferences among thousands of songs.

Bulbul allows its users to explore music from many different artists and genres around the world. Since also it is developed using English as the user interface language, its impact could be global.

Security of user data is one of our main concerns. In order to keep user passwords safe, our backend algorithm uses bcrypt algorithm with a secret key to convert them from plain text to irreversible hashes. This protects the actual user passwords in a possible database leak scenario.

# 6. Final Status of the Project

The system is functional as it is in the current state however, some user information gathering procedures are not fully implemented. Currently recommendations are generated based on the ratings the user will give on the opening page of the application. The recommendations will be improved once the user starts using the application and rates more songs when the data gathering procedures are at their final form.

# 7. Software and Hardware Systems

Bulbul essentially runs on two hardware systems:

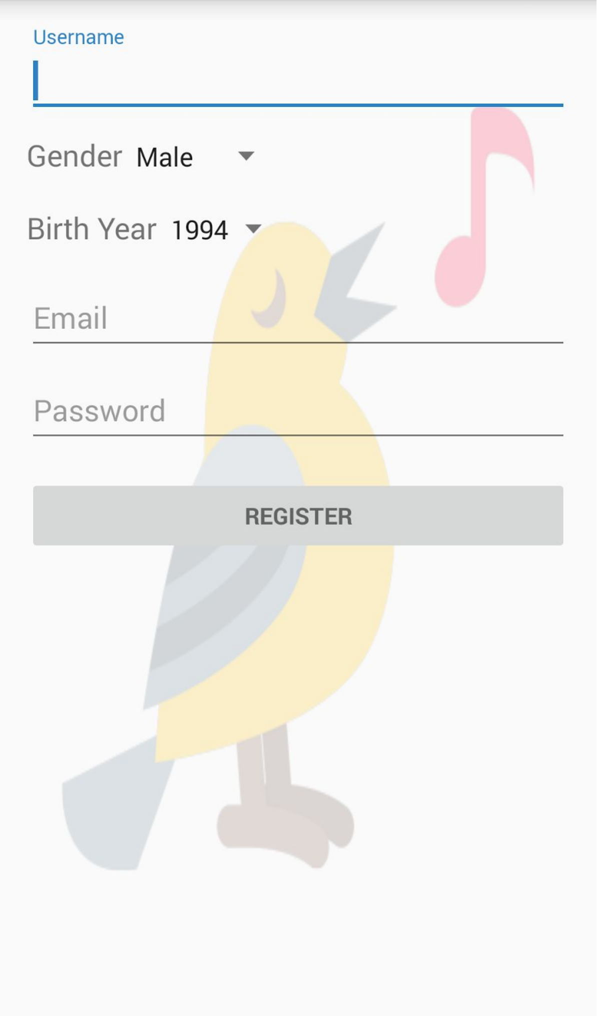
## 7.1. Smartphone

For users to use the Bulbul application they need a smartphone. Each user can access their personal recommendations this way. The software which the user will use is the android application on the smartphone. This application registers the user data to the server side while also providing an interface to the user to interact with the system. The application will also allow the user to listen to music and rate them.

## 7.2. Server

The data collected from users are stored on the server side which is actually a cloud service provided by Digital Ocean. The processing of the data and the recommendations are generated on this platform. The cloud server runs the code for the recommendation algorithm and also hosts the Neo4j database.

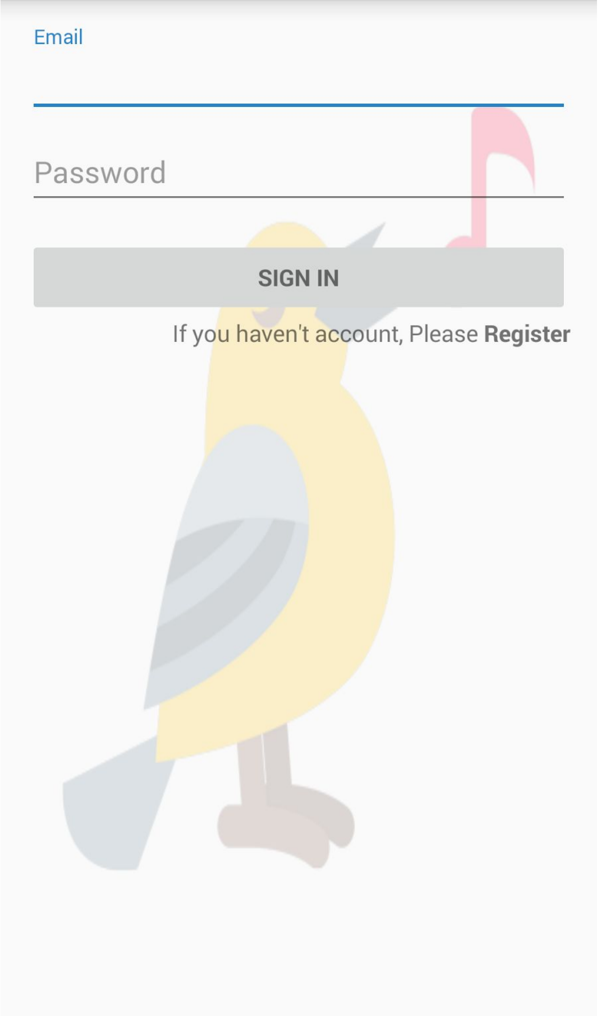
# 8. User Manual



Gender Male

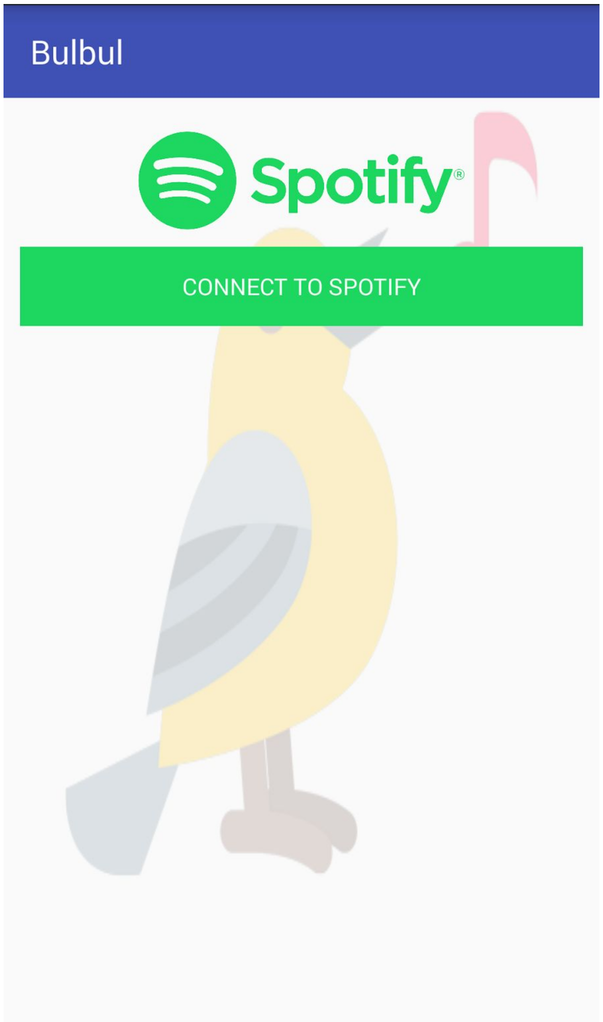
Birth Year 1994

Register Screen

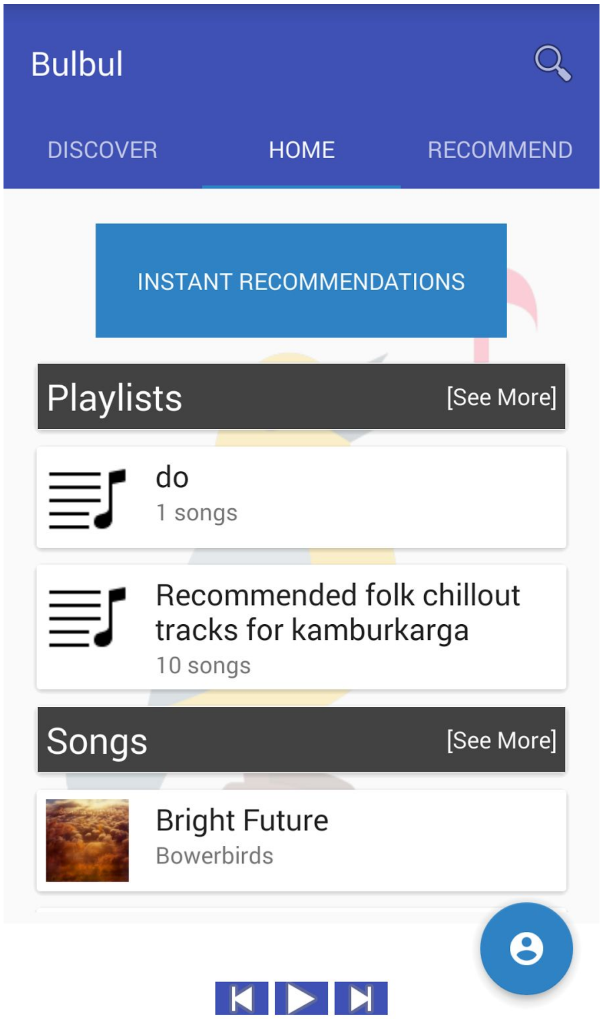


If you haven't account, Please **Register**

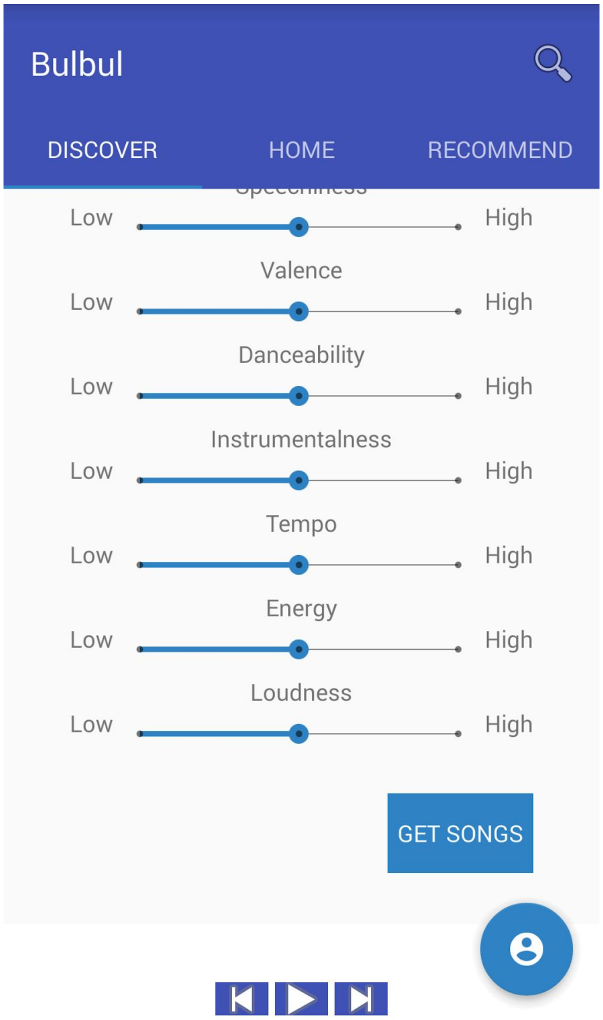
Login Screen



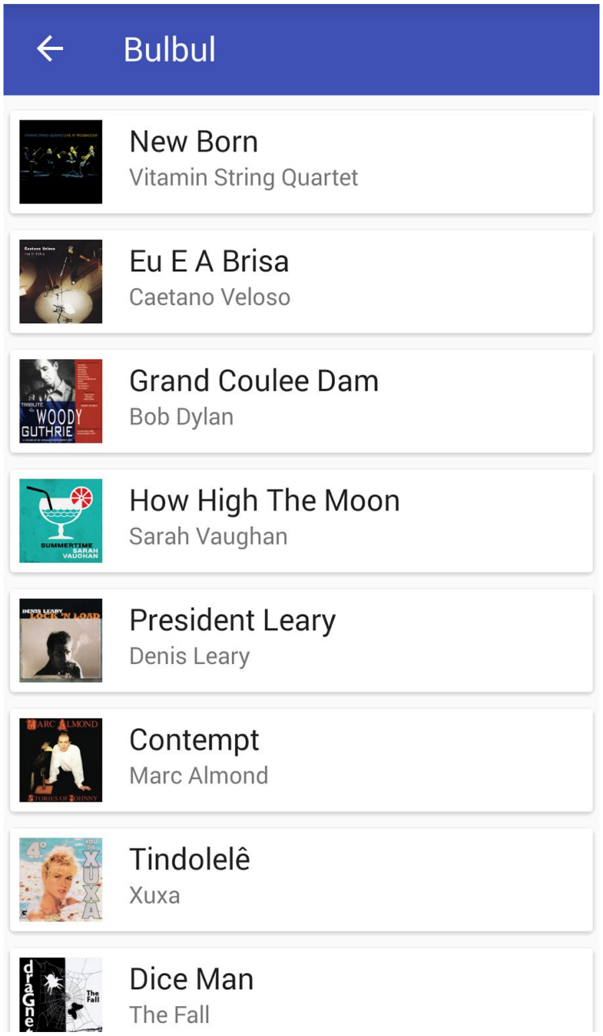
Spotify Connection Screen



Home Screen



Discover Screen



New Born
Vitamin String Quartet

Eu E A Brisa
Caetano Veloso

Grand Coulee Dam
Bob Dylan

How High The Moon
Sarah Vaughan

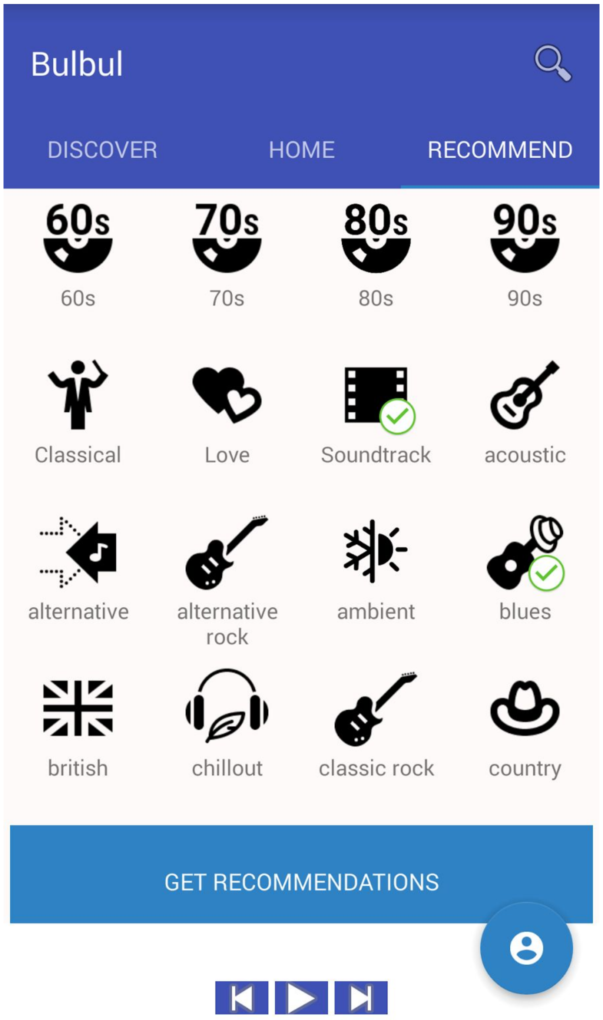
President Leary
Denis Leary

Contempt
Marc Almond

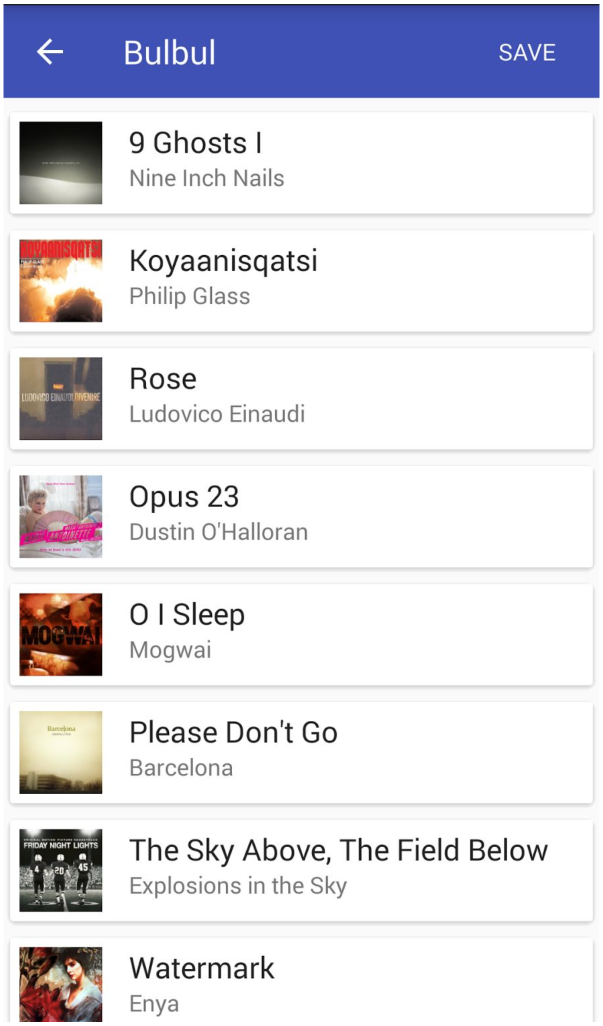
Tindolelê
Xuxa

Dice Man
The Fall

Discovery Results Screen



Recommendation Screen



9 Ghosts I
Nine Inch Nails

Koyaanisqatsi
Philip Glass

Rose
Ludovico Einaudi

Opus 23
Dustin O'Halloran

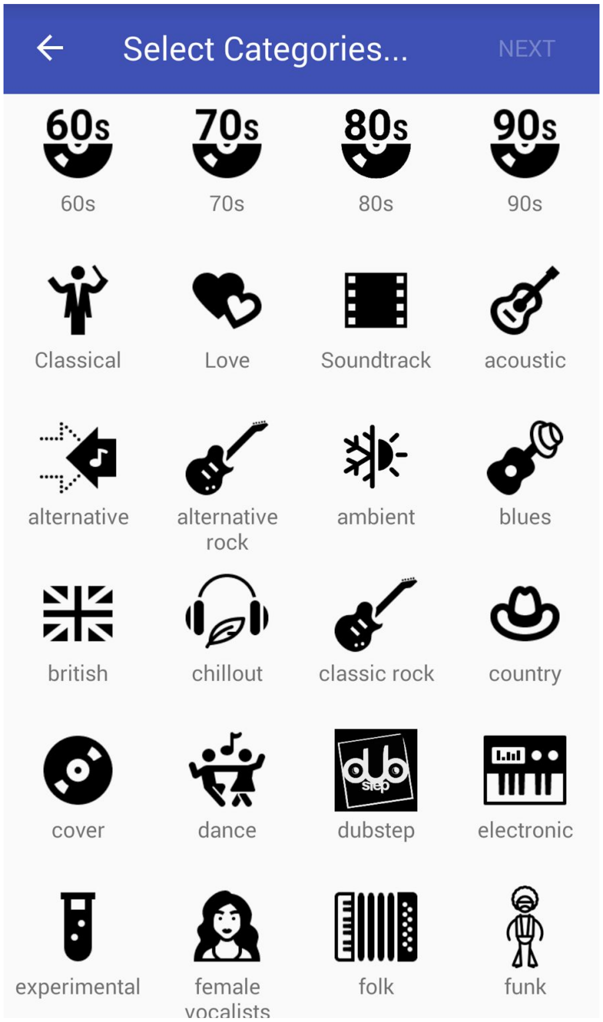
O I Sleep
Mogwai

Please Don't Go
Barcelona

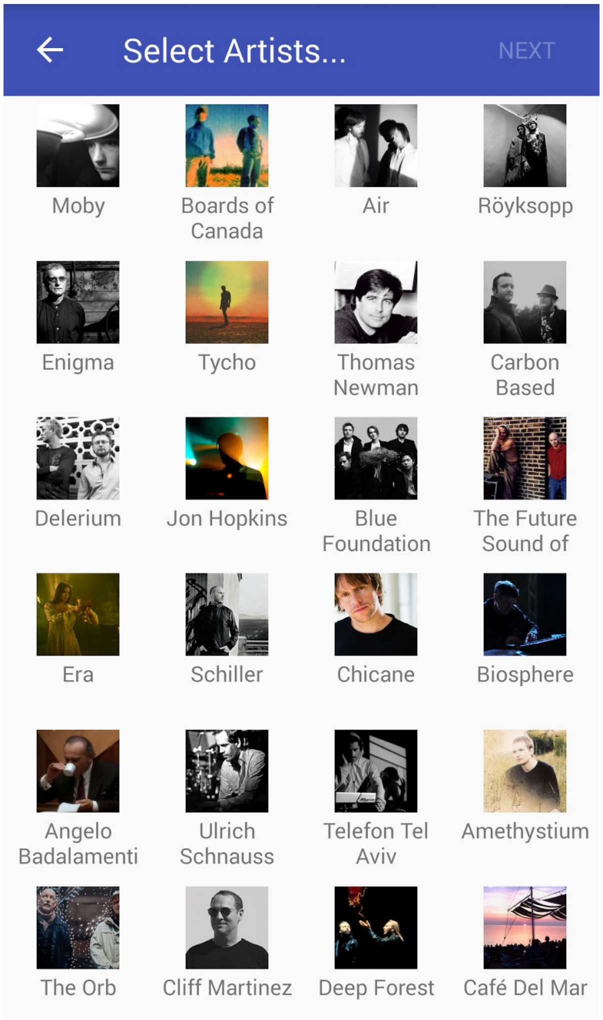
The Sky Above, The Field Below
Explosions in the Sky

Watermark
Enya

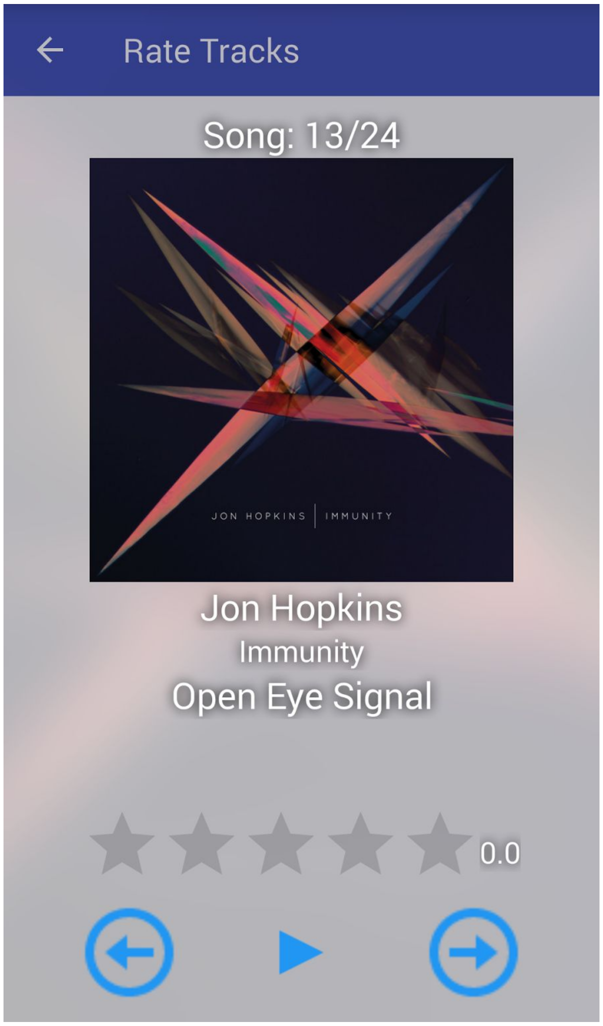
Recommendation Results Screen



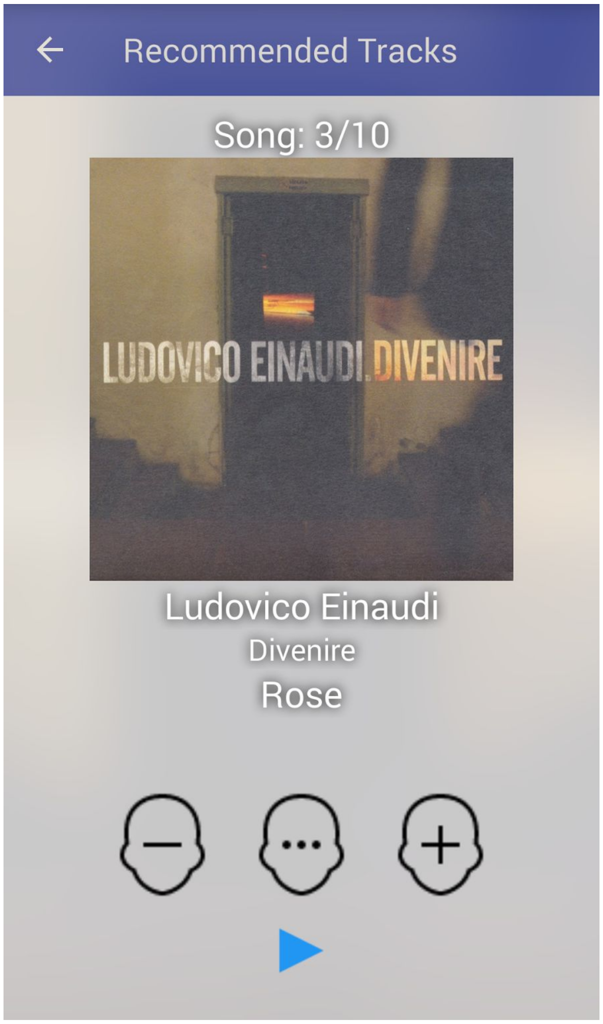
Instant Recommendation
Category Selection Screen



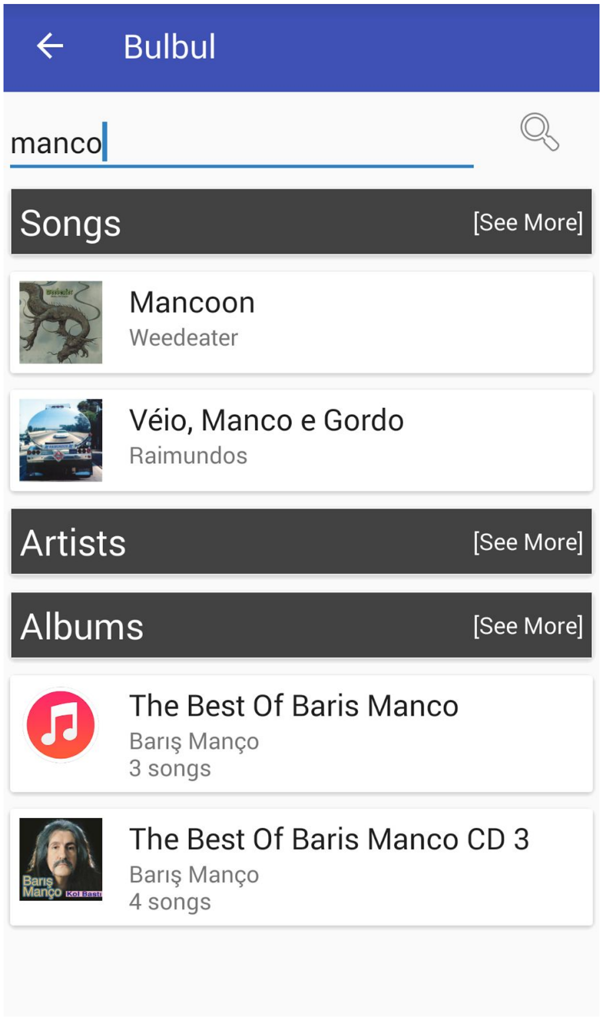
Instant Recommendation
Artist Selection Screen



Instant Recommendation
Track Rating Screen



Instant Recommendation
Result Screen



manco

Songs [See More]

Mancoon
Weedeater

Véio, Manco e Gordo
Raimundos

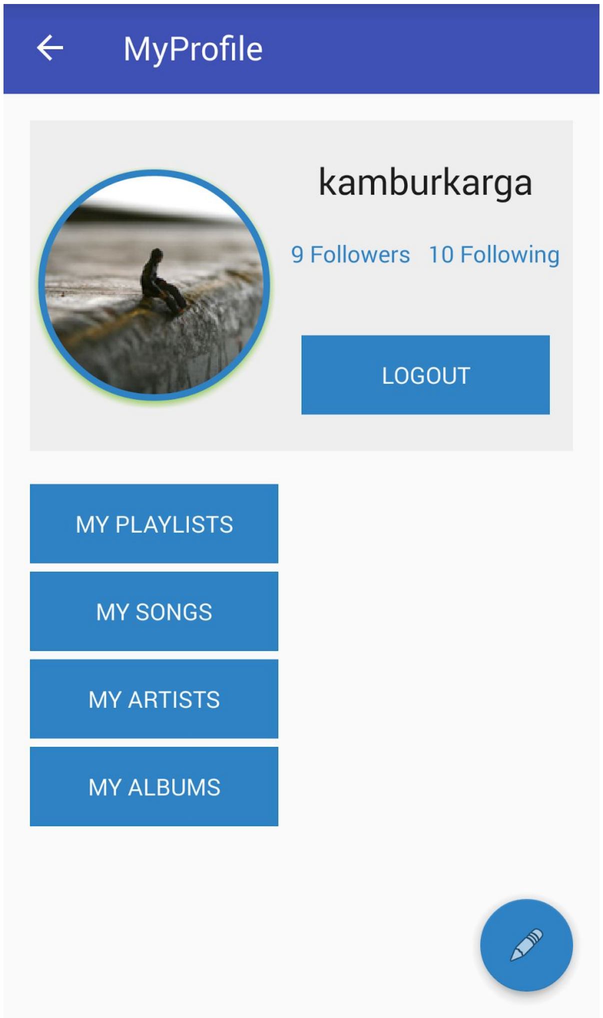
Artists [See More]

Albums [See More]

The Best Of Baris Manco
Barış Manço
3 songs

The Best Of Baris Manco CD 3
Barış Manço
4 songs

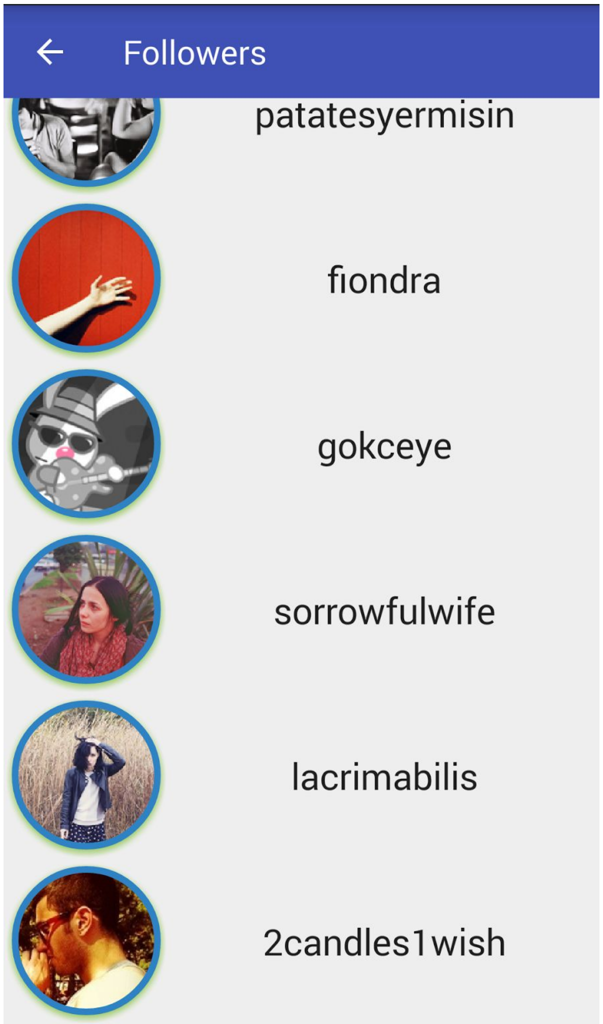
Search Screen



kamburkarga

9 Followers 10 Following

Profile Screen



Followers

patatesyermisin

fiondra

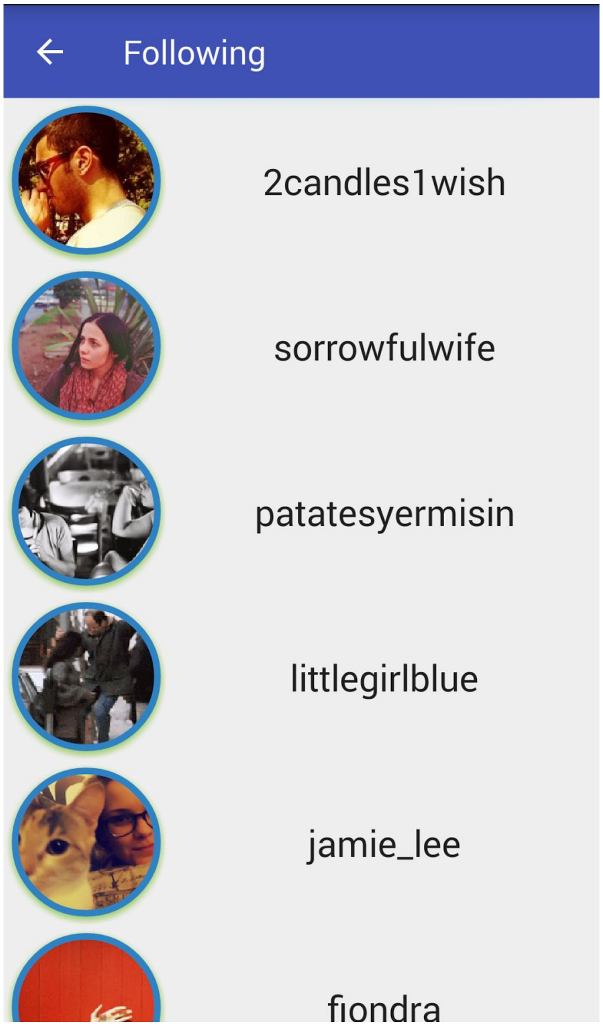
gokceye

sorrowfulwife

lacrimabilis

2candles1wish

Followers Screen



Following

2candles1wish

sorrowfulwife

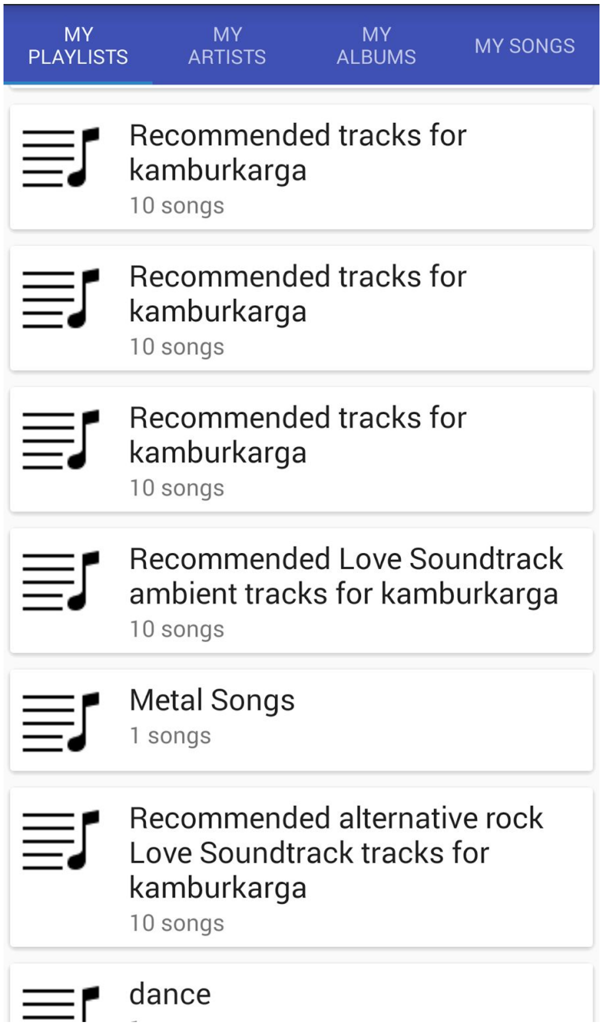
patatesyermisin

littlegirlblue

jamie_lee

fiondra

Following Screen



Recommended tracks for kamburkarga
10 songs

Recommended tracks for kamburkarga
10 songs

Recommended tracks for kamburkarga
10 songs

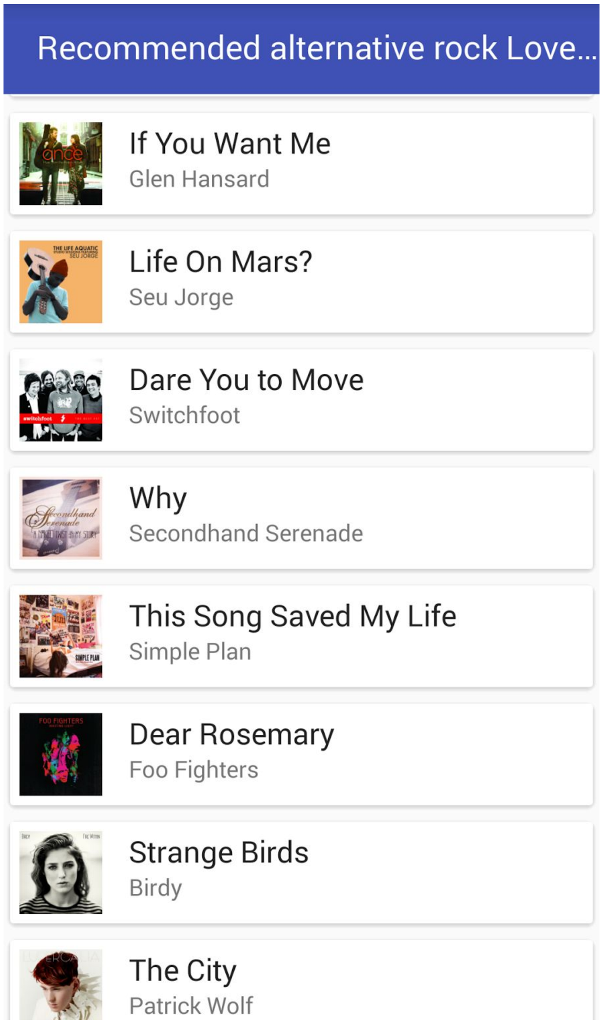
Recommended Love Soundtrack ambient tracks for kamburkarga
10 songs

Metal Songs
1 songs

Recommended alternative rock Love Soundtrack tracks for kamburkarga
10 songs

dance

My Playlists Screen



Recommended alternative rock Love...

If You Want Me
Glen Hansard

Life On Mars?
Seu Jorge

Dare You to Move
Switchfoot

Why
Secondhand Serenade

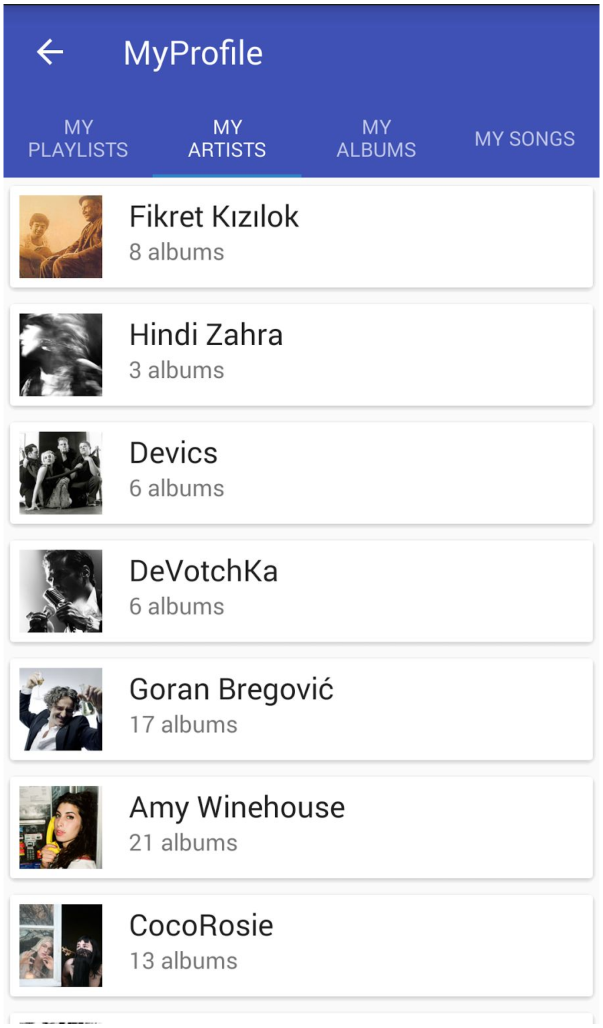
This Song Saved My Life
Simple Plan

Dear Rosemary
Foo Fighters

Strange Birds
Birdy

The City
Patrick Wolf

Playlist Screen



Fikret Kızılok
8 albums

Hindi Zahra
3 albums

Devics
6 albums

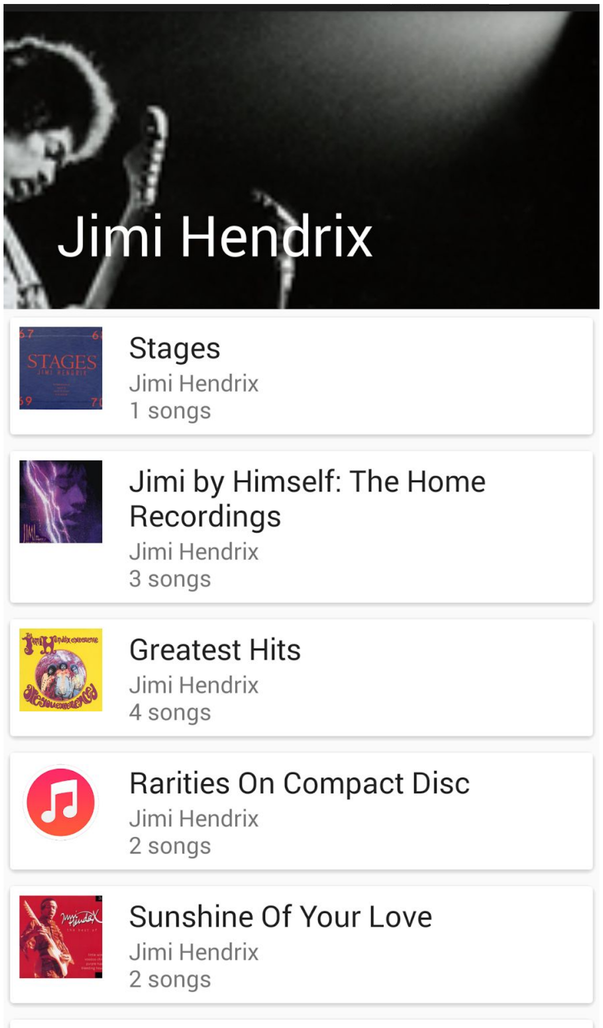
DeVotchKa
6 albums

Goran Bregović
17 albums

Amy Winehouse
21 albums

CocoRosie
13 albums

My Artists Screen



Stages
Jimi Hendrix
1 songs

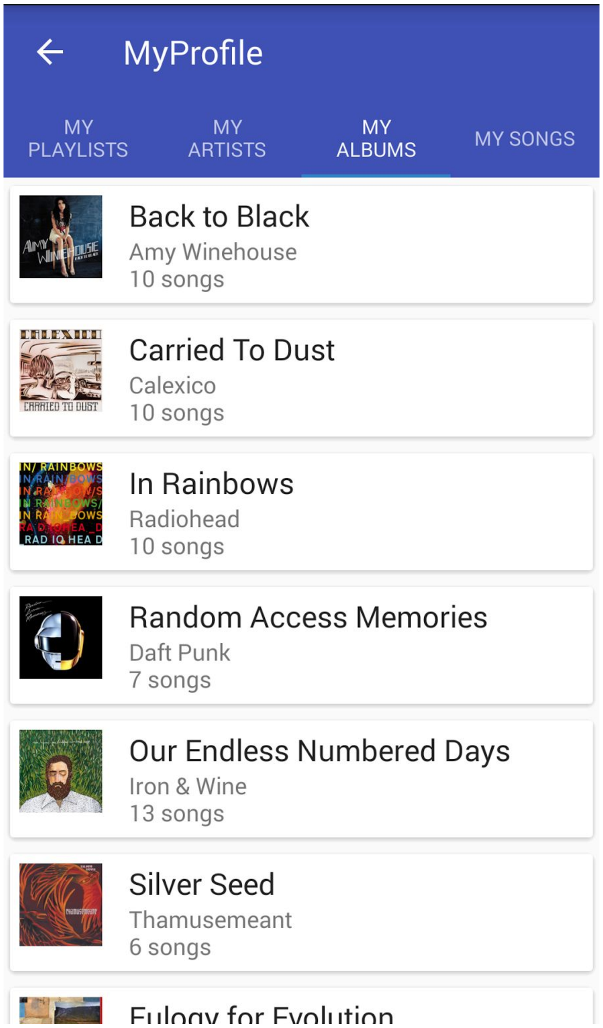
Jimi by Himself: The Home Recordings
Jimi Hendrix
3 songs

Greatest Hits
Jimi Hendrix
4 songs

Rarities On Compact Disc
Jimi Hendrix
2 songs

Sunshine Of Your Love
Jimi Hendrix
2 songs

Artist Screen



MyProfile

Back to Black
Amy Winehouse
10 songs

Carried To Dust
Calexico
10 songs

In Rainbows
Radiohead
10 songs

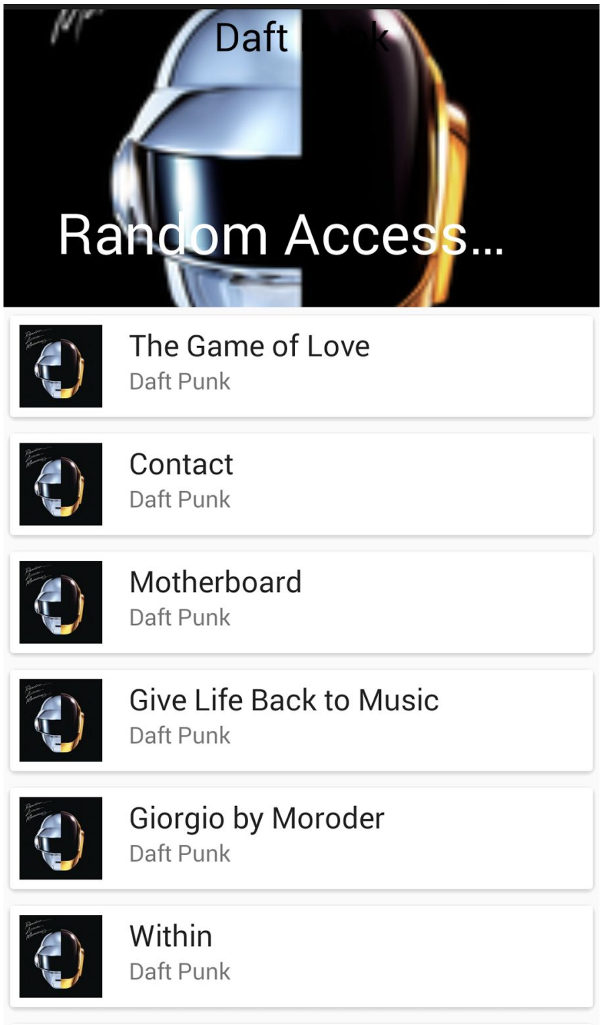
Random Access Memories
Daft Punk
7 songs

Our Endless Numbered Days
Iron & Wine
13 songs

Silver Seed
Thamusemeant
6 songs

Eulogy for Evolution

My Albums Screen



Daft k

Random Access...

The Game of Love
Daft Punk

Contact
Daft Punk

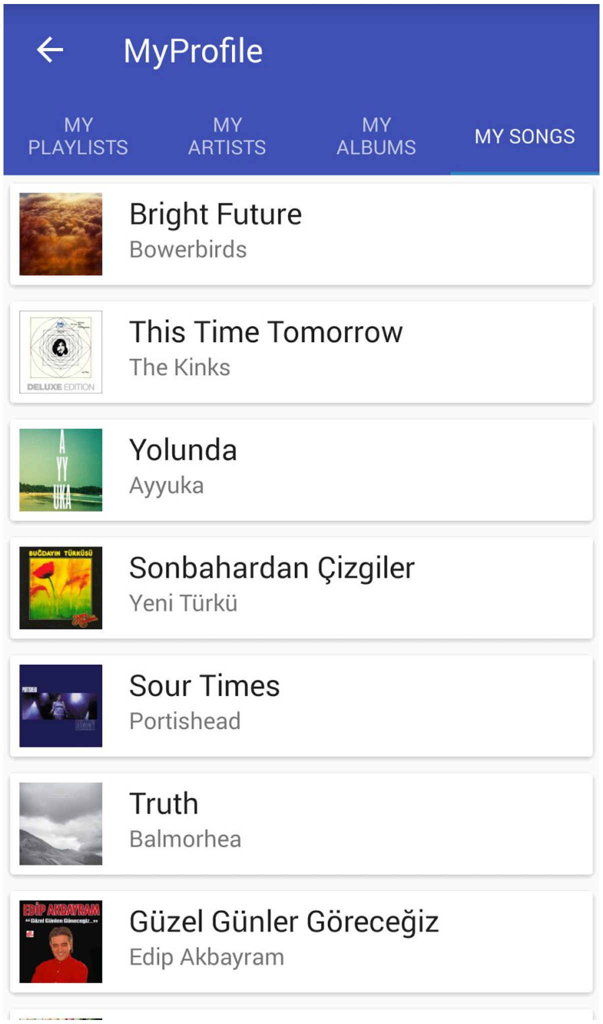
Motherboard
Daft Punk

Give Life Back to Music
Daft Punk

Giorgio by Moroder
Daft Punk

Within
Daft Punk

Album Screen



Bright Future
Bowerbirds

This Time Tomorrow
The Kinks

Yolunda
Ayyuka

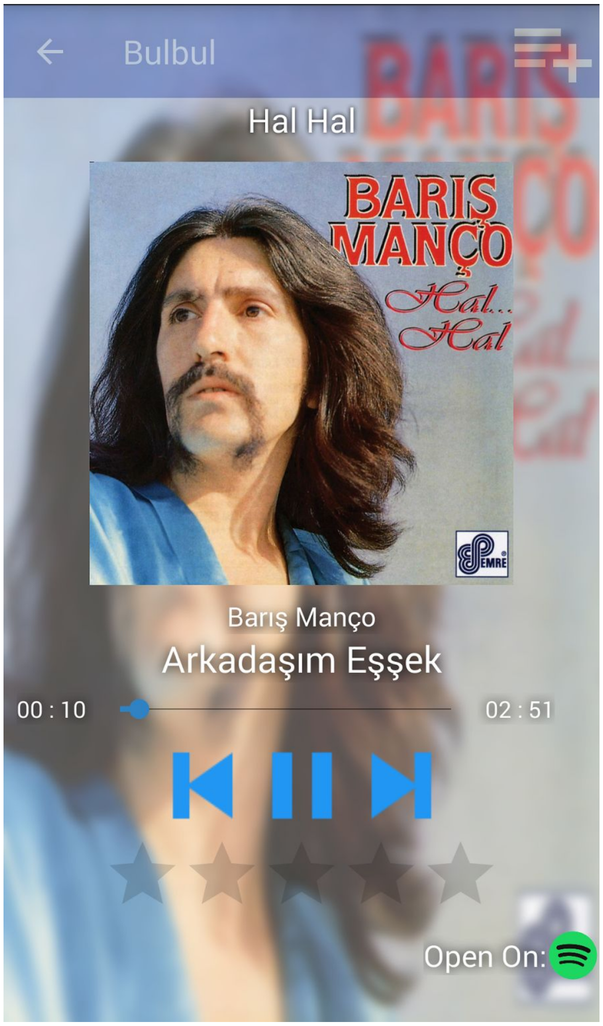
Sonbahardan Çizgiler
Yeni Türkü

Sour Times
Portishead

Truth
Balmorhea

Güzel Günler Göreceğiz
Edip Akbayram

My Songs Screen



Hal Hal

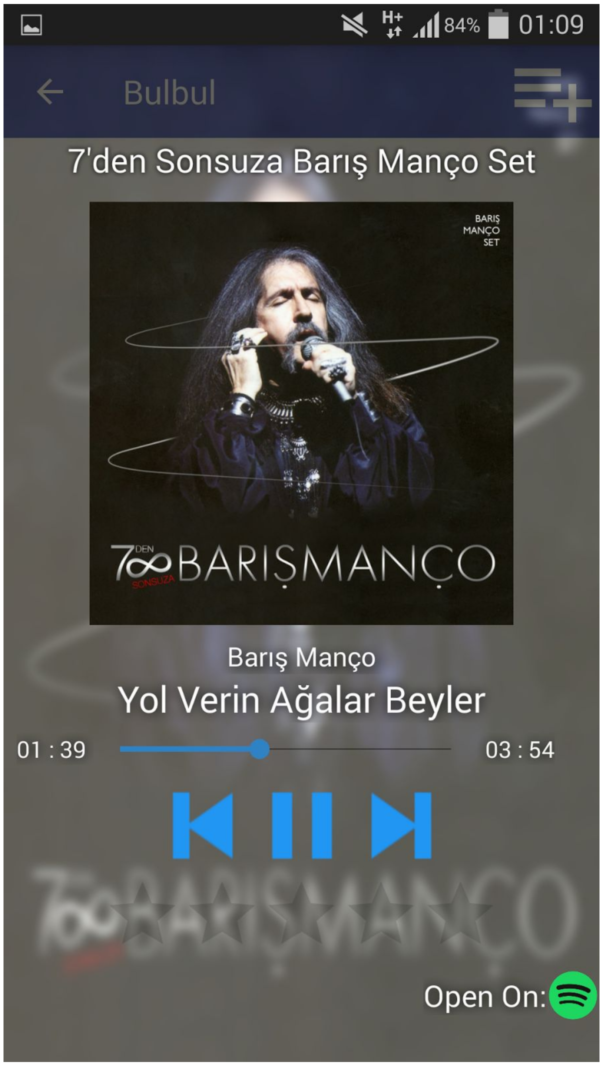
Barış Manço

Arkadaşım Eşşek

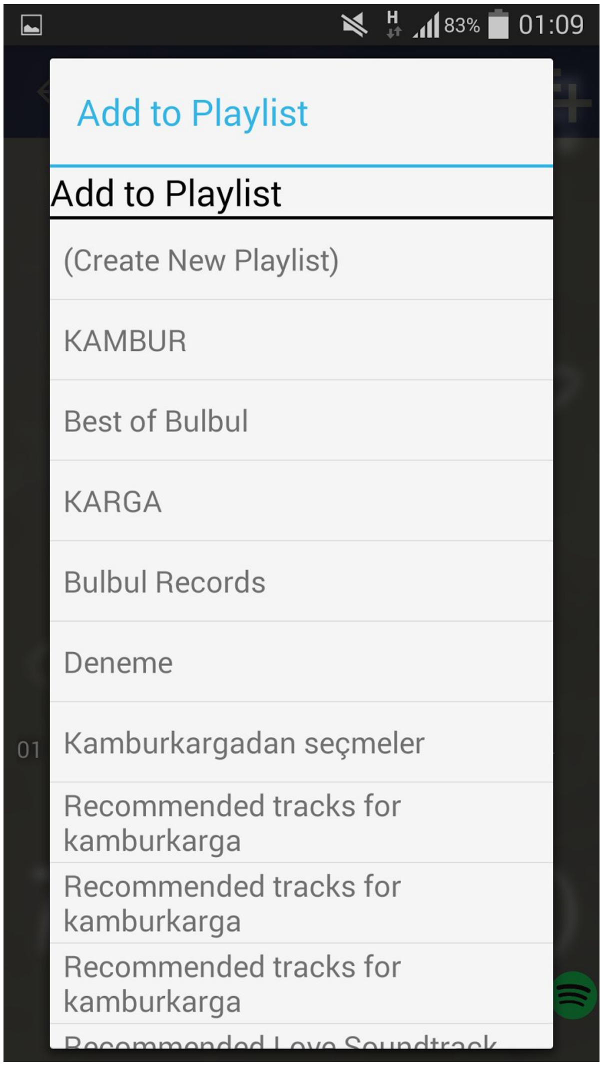
00 : 10

02 : 51

Streaming Screen



Streaming Screen



Add to Playlist

Add to Playlist

- (Create New Playlist)
- KAMBUR
- Best of Bulbul
- KARGA
- Bulbul Records
- Deneme
- Kamburkargadan seçmeler
- Recommended tracks for kamburkarga
- Recommended tracks for kamburkarga
- Recommended tracks for kamburkarga
- Recommended Love Soundtrack

Add To Playlist Screen

# References:

[1]"Fabric - App Development Platform for teams", Fabric.io, 2017. [Online]. Available: https://fabric.io/.
[2]"Introduction | Apollo Android Guide", dev.apollodata.com, 2017. [Online]. Available: http://dev.apollodata.com/android/.
[3]"DigitalOcean: Cloud computing designed for developers", DigitalOcean, 2017. [Online]. Available: https://www.digitalocean.com/.
[4]"Neo4j, the world's leading graph database - Neo4j Graph Database", Neo4j Graph Database, 2017. [Online]. Available: https://neo4j.com/
[5]"GraphQL: A query language for APIs.", Graphql.org, 2017. [Online]. Available: http://graphql.org/
[6]T. Otwell, "Laravel - The PHP Framework For Web Artisans", Laravel.com, 2017. [Online]. Available: https://laravel.com/
[7]"Redis", Redis.io, 2017. [Online]. Available: https://redis.io/