

# Chatbot Song Recommender System

## Overview

### Objective

In this project, we would be combining multiple services and open-source tools to make a Chatbot that recommends songs based on the tone of the conversation which the user is having with the chatbot.

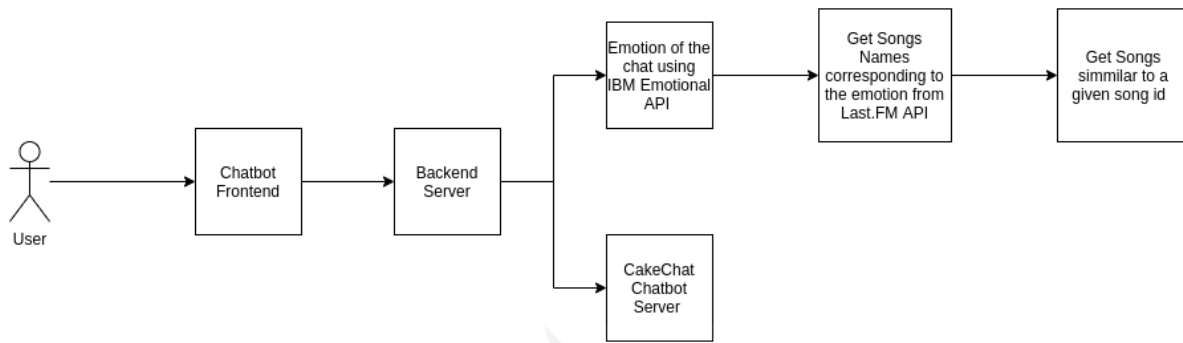
### Project Context

The purpose of chat bots is to support and scale business teams in their relations with customers. It could live in any major chat applications like Facebook Messenger, Slack, Telegram, Text Messages, etc. Chatbot applications streamline interactions between people and services, enhancing customer experience. At the same time, they offer companies new opportunities to improve the customers engagement process and operational efficiency by reducing the typical cost of customer service. This project is focussed on building a custom chatbot that will be your fundamental step of the learning curve of building your own professional chatbots.

But you must be tired of the weird chat bots out there in the world which are made for mainly business purposes? In this project, we would be building an extensive Chatbot service, to which you can talk to. And talking to a chatbot wouldn't be business-driven. It would just be casual conversations. Further, on top of it, the chatbot would also be recommending songs to the user based on the tone of the user. This song recommendation feature employs the use of [Last.fm API](#), very much similar to the popular Spotify API. Also for tone/emotion analysis of the conversation we will be using the [IBM Tone Analyzer API](#).

Collaborating with these types of APIs is very much critical as in today's world the popular chatbots do much more than simply having a data-driven conversation; to supplement additional user-oriented features. Also the reason to choose python to build the chatbot is because python boasts a wide array of open-source libraries for chatbots, including scikit-learn and TensorFlow. It is great for small data sets and more simple analyses; also Python's libraries are much more practical.

### Product Architecture



## High-Level Approach

- User starts the conversation
- Emotional Analysis of the conversation is done using the IBM Emotional API
- Get the reply to the conversation from the Cakechat Chatbot
- Based on the Emotion which the app perceives, top songs are retrieved using Last.fm songs API
- If a user listens to a particular song for some time, a similar song would be recommended to the user using Last.fm API.

## Primary goals

- Setting up an open-source project locally and handling the errors being faced
- Using multiple services to build up a new service over them.
- Having a real-world chatbot, to which you can literally chat like you chatting to a real person and enjoying the music recommended by the system.

## Task 1

### Setting up the CakeChat Chatbot Server locally

Since Chatbot is the main part of the project, we would be setting up the Chatbot first. This part alone would give you a strong understanding of the project.

In this milestone, you need to install Cakechat Chatbot from [this GitHub Repository](#).

## Requirements

- Create a virtual environment of python-3.
- Fork and Clone the GitHub repository onto your local machine.
- Install the dependencies as specified [here](#).

- Load the pre-trained model by following the steps specified [here](#).
- Run and test the Cakechat Server as specified [here](#)

## References

- [Cakechat Github Repository](#)
- [Installing the dependencies](#)
- [Loading the pre-trained model](#)
- [Running and Testing the Cakechat Server](#)

## Tip

- If you see the following error:-

```
ModuleNotFoundError: No module named 'boto3'
```

Install the python dependency boto3 by using the following command:-

```
pip install boto3
```

- If you see the following error:-

```
AttributeError: 'str' object has no attribute 'decode'
```

have a look [here](#).

## Expected Outcome

- By the end, you will be having a running Cakechat Chatbot Server on your system and you would be able to get a response based on a conversation like shown below:-

```
python tools/test_api.py -f 127.0.0.1 -p 8080 \  
> -c "Hi, Eddie, what's up?" \  

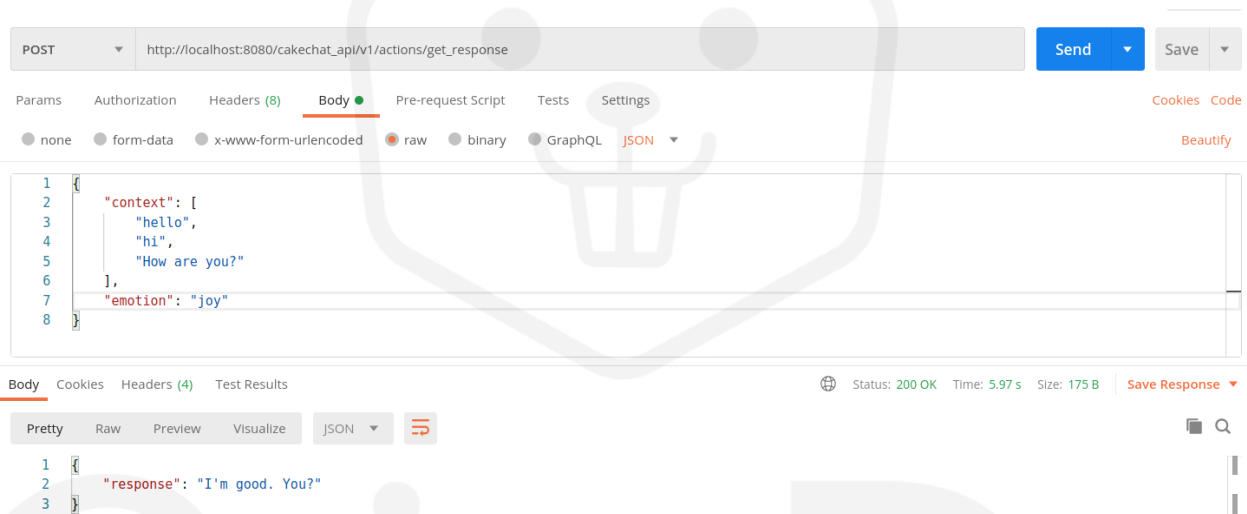
```

```
> -c "Not much, what about you?" \  
> -c "Fine, thanks. Are you going to the movies tomorrow?"
```

Using TensorFlow backend.

```
{'response': "Yes. I was going to go but didn't know you're coming!"}
```

- Further, on testing it using Postman, the output looks like this:-



## Task 2

### Setting Up the IBM Tone Analyzer API

In this milestone, we would be setting up the IBM Tone Analyzer API so that we can analyze the tone of conversation (emotions). We are using an API here as we don't have that enough data, computational power and time to create our own model API. This milestone will make

you realise why to prefer using open-source APIs rather than creating your own models each time.

## Requirements

- Check [this](#) website to have an idea of what IBM Tone Analyzer API would be doing.
- Create an account on [IBM Cloud](#) (It's free)
- Enable the Tone Analyzer Service for your account from [here](#).
- Try running the Python code for analyzing the conversation from [here](#) and don't forget to replace {apikey} and {url} with the apikey and url you received by enabling Tone Analyser Service for your account.

## References

- [IBM Tone Analyzer API demo](#)
- [Python Code to use IBM Tone API](#)

## Tip

- If you see any error like

```
jwt.exceptions.DecodeError: It is required that you pass in a value for the "algorithms" argument when calling decode().
```

Then have a look [here](#) and install the following dependency

```
pip install PyJWT==1.7.1
```

## Expected Outcome

- After running the code, tone Analyser Service, you would have the analysis of the text on your system which looks similar to this. Here as the top is the tone of the entire app and below is the tone of every sentence.



```
{
  "document_tone": {

    "tones": [
      {
        "score": 0.6165,
        "tone_id": "sadness",
        "tone_name": "Sadness"
      },

      {
        "score": 0.829888,
        "tone_id": "analytical",
        "tone_name": "Analytical"
      }
    ]
  },

  "sentences_tone": [
    {
      "sentence_id": 0,
      "text": "Team, I know that times are tough!",
      "tones": [

        {
          "score": 0.801827,
          "tone_id": "analytical",
          "tone_name": "Analytical"
        }
      ]
    },

    {
      "sentence_id": 1,
      "text": "Product sales have been disappointing for the past three
quarters.",

      "tones": [
        {
          "score": 0.771241,
          "tone_id": "sadness",
          "tone_name": "Sadness"
        }
      ]
    }
  ]
}
```

```
    },  
  
    {  
      "score": 0.687768,  
      "tone_id": "analytical",  
      "tone_name": "Analytical"  
    }  
  ]  
},  
  
{  
  "sentence_id": 2,  
  "text": "We have a competitive product, but we need to do a better  
job of selling it!",  
  
  "tones": [  
    {  
      "score": 0.506763,  
      "tone_id": "analytical",  
      "tone_name": "Analytical"  
    }  
  ]  
}  
]
```

So, now for any given phrase or sentence, you can analyze the tone of that. We would be using this code ahead to analyze the tone of the conversation.

### Task 3

#### Setting up Last.fm Songs API and testing it

In this milestone, we would be setting up the Last.fm Songs API so that we can recommend some songs to the user based on the tone/emotion of the user. We are using an API here as we don't have enough data, computational power and time to scrape the web for songs based on the specific information of tones we have extracted previously.

## Requirements

- Create API account on Last.fm from [here](#) and get the API\_KEY
- Test the following API's features with the help of your API\_KEY
  - Getting top 5 songs of a particular tag
  - Getting similar songs related to a song

## References

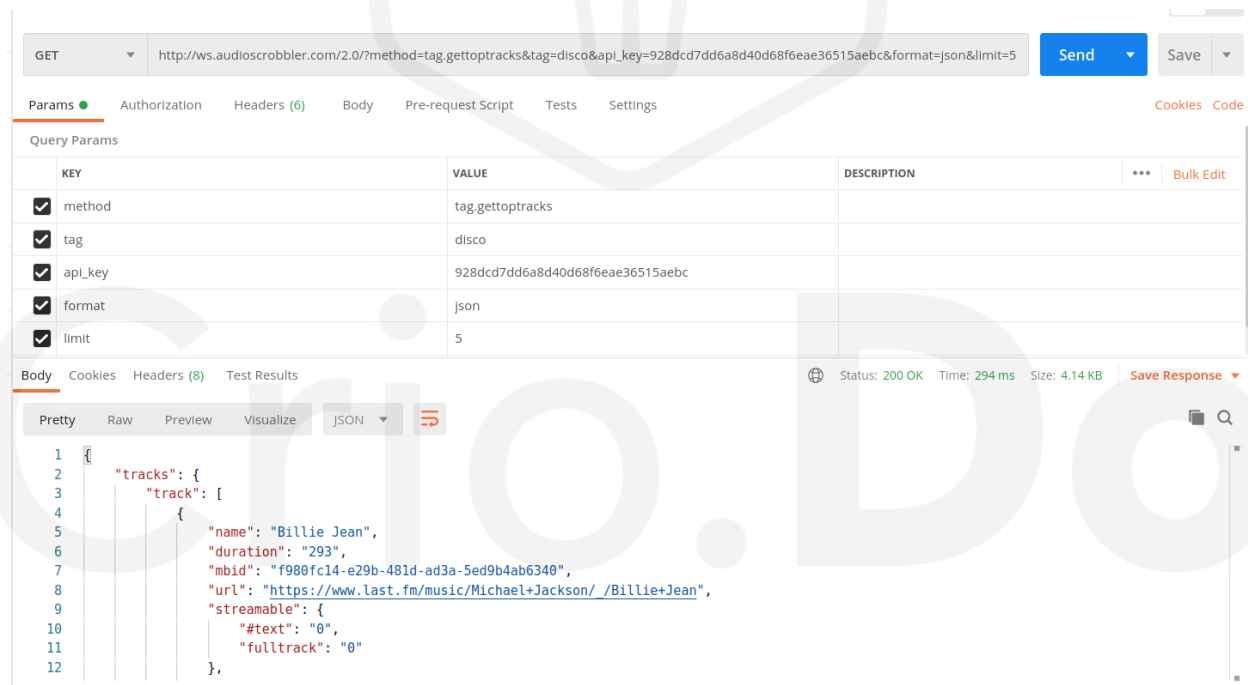
- [Getting top 5 songs of a particular tag](#)
- [Getting similar songs related to a song](#)

## Note

Work with JSON API's only and not XML API

## Expected Outcome

- By the end of this milestone, you would have seen the response of both the above-specified API's
- Getting top 5 songs of a particular tag



The screenshot displays a REST client interface with the following details:

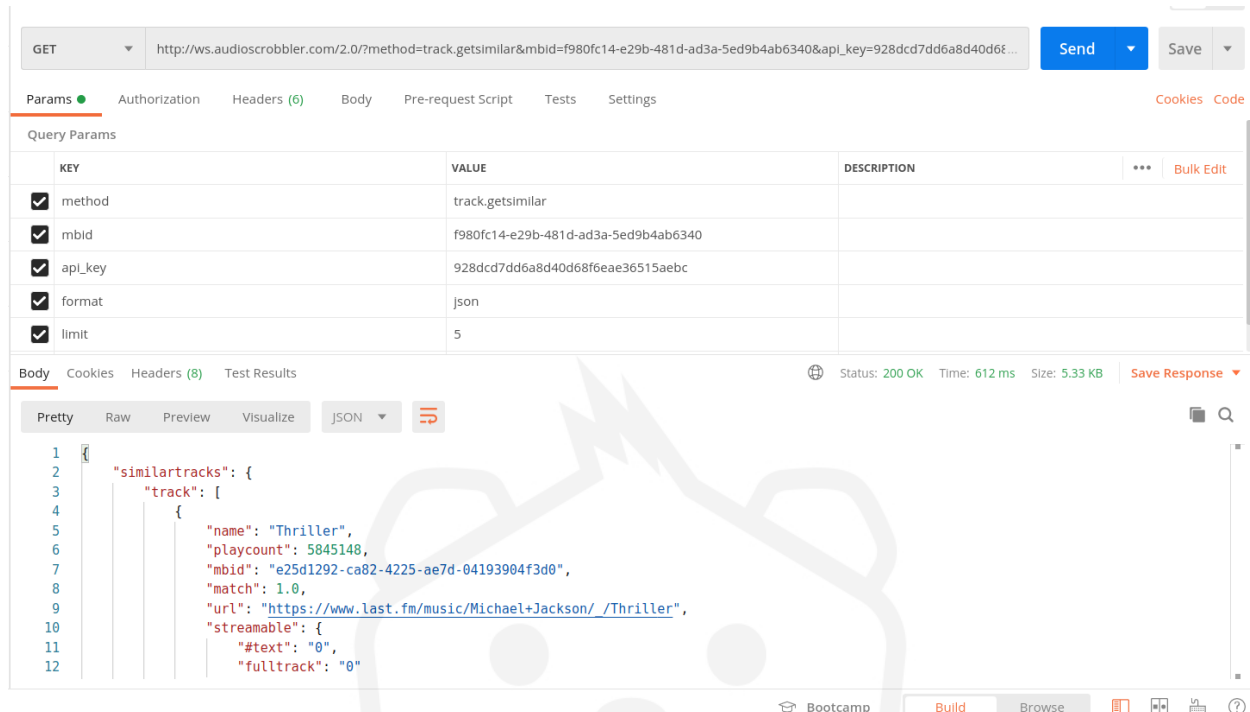
- Method:** GET
- URL:** `http://ws.audioscrobbler.com/2.0/?method=tag.gettoptracks&tag=disco&api_key=928dcd7dd6a8d40d68f6eae36515aebc&format=json&limit=5`
- Params:** Authorization, Headers (6), Body, Pre-request Script, Tests, Settings
- Query Params Table:**

KEY	VALUE	DESCRIPTION
method	tag.gettoptracks	
tag	disco	
api_key	928dcd7dd6a8d40d68f6eae36515aebc	
format	json	
limit	5	
- Status:** 200 OK, Time: 294 ms, Size: 4.14 KB
- Response Body (JSON):**

```
{
  "tracks": {
    "track": [
      {
        "name": "Billie Jean",
        "duration": "293",
        "mbid": "f980fc14-e29b-481d-ad3a-5ed9b4ab6340",
        "url": "https://www.last.fm/music/Michael+Jackson/_/Billie+Jean",
        "streamable": {
          "#text": "0",
          "fulltrack": "0"
        }
      }
    ]
  }
}
```

- Get similar songs related to a given song id





GET [http://ws.audioscrobbler.com/2.0/?method=track.getsimilar&mbid=f980fc14-e29b-481d-ad3a-5ed9b4ab6340&api\\_key=928dcd7dd6a8d40d6f...](http://ws.audioscrobbler.com/2.0/?method=track.getsimilar&mbid=f980fc14-e29b-481d-ad3a-5ed9b4ab6340&api_key=928dcd7dd6a8d40d6f...) Send Save

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies Code

Query Params

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> method	track.getsimilar	
<input checked="" type="checkbox"/> mbid	f980fc14-e29b-481d-ad3a-5ed9b4ab6340	
<input checked="" type="checkbox"/> api_key	928dcd7dd6a8d40d6f6eae36515aebc	
<input checked="" type="checkbox"/> format	json	
<input checked="" type="checkbox"/> limit	5	

Body Cookies Headers (8) Test Results Status: 200 OK Time: 612 ms Size: 5.33 KB Save Response

Pretty Raw Preview Visualize JSON

```

1 {
2   "similartracks": {
3     "track": [
4       {
5         "name": "Thriller",
6         "playcount": 5845148,
7         "mbid": "e25d1292-ca82-4225-ae7d-04193904f3d0",
8         "match": 1.0,
9         "url": "https://www.last.fm/music/Michael+Jackson/_/Thriller",
10        "streamable": {
11          "#text": "0",
12          "fulltrack": "0"

```

## Tip

Try using `mbid` instead of just using track name and artist for finding songs similar to a given song.

## Task 4

### Understanding the overall process involved in the chatbot's working

Once we are done with this milestone, we would be having a very good understanding of the complete project architecture. Further, after this, you can take this project in a new direction after this milestone. Completing this milestone is critical as there are a lot of components that need to be integrated together to complete the chatbot; having a clear sense of product architecture is needed here.

- Have a look at the High-Level approach which we briefed earlier

#### High-Level Approach

- User starts the conversation
- Emotional Analysis of the conversation is done using the IBM Emotional API
- Get the reply to the conversation from the Cakechat Chatbot
- Based on the Emotion which we got, top songs are retrieved using Last.fm songs API

- If a user listens to a particular song ... a similar song to that song would be recommended to the user using Last.fm API.

We would be implementing this approach manually to get started right away by building the chatbot in the next milestone.

## Requirements

- Let's start the conversation with a good mood tone:  
Hey there!! What's up? How's the day?
- Now let's send this text for Emotional Analysis. And we get the following response:

```
{
  "document_tone": {
    "tones": [
      {
        "score": 0.66525,
        "tone_id": "joy",
        "tone_name": "Joy"
      }
    ]
  },
  "sentences_tone": [
    {
      "sentence_id": 0,
      "text": "Hey there!! What's up?",
      "tones": []
    },
    {
      "sentence_id": 1,
      "text": "How's the day?",
      "tones": []
    }
  ]
}
```

- Now send the request to the chatbot server with the emotion of the chatbot using the following mapping:

```
{
  "anger": "anger",
  "fear": "sadness",
  "joy": "joy",
  "sadness": "sadness",
  "analytical": "neutral",
  "confident": "neutral",
  "tentative": "neutral"
}
```

Why this mapping? Because the Cakechat chatbot supports conversation in limited emotions only.

Just wondering... how can you have an analytical conversation? ... but still, you can have an analytical tone ... Therefore we are mapping Analytical tone to Neutral conversation.

So, using the above mapping we would be talking to the server.

- Now, send the request to the cakechat server with the conversation and the emotion of the chat.

Request:

```
{
  "context": [
    "Hey there!! What's up? How's the day?"
  ],
  "emotion": "joy"
}
```

Response:

```
{    "response": "It's been good. How about you?"    }
```

As you can see we got the response. So, you can continue further with the conversation appended in the context variable.

- Now, it's time to get a song recommendation. Have a look and analyze the number of songs of a particular tag using last.fm tag.gettoptracks API.

```
{  "tracks": {    "track": [      {...},      {...}    ],    "@attr": {      "tag": "analytical",      "page": "1",      "perPage": "5",      "totalPages": "1",      "total": "2"    }  } }
```

You would have seen here also that we were only able to find 2 songs corresponding to the `analytical` tag. So, instead of using the tags returned by the Emotional API, we can map them using the mapping defined earlier above.

- Now, once a user would click on a particular song, we also need to recommend similar songs to that particular song. This can also be done using the last.fm's `track.getsimilar` API using the track and the artist of the song.

## Expected Outcome

By now, you have a complete system and a good understanding of the project's flow, which is satisfied by the above requirements.

## Task 5

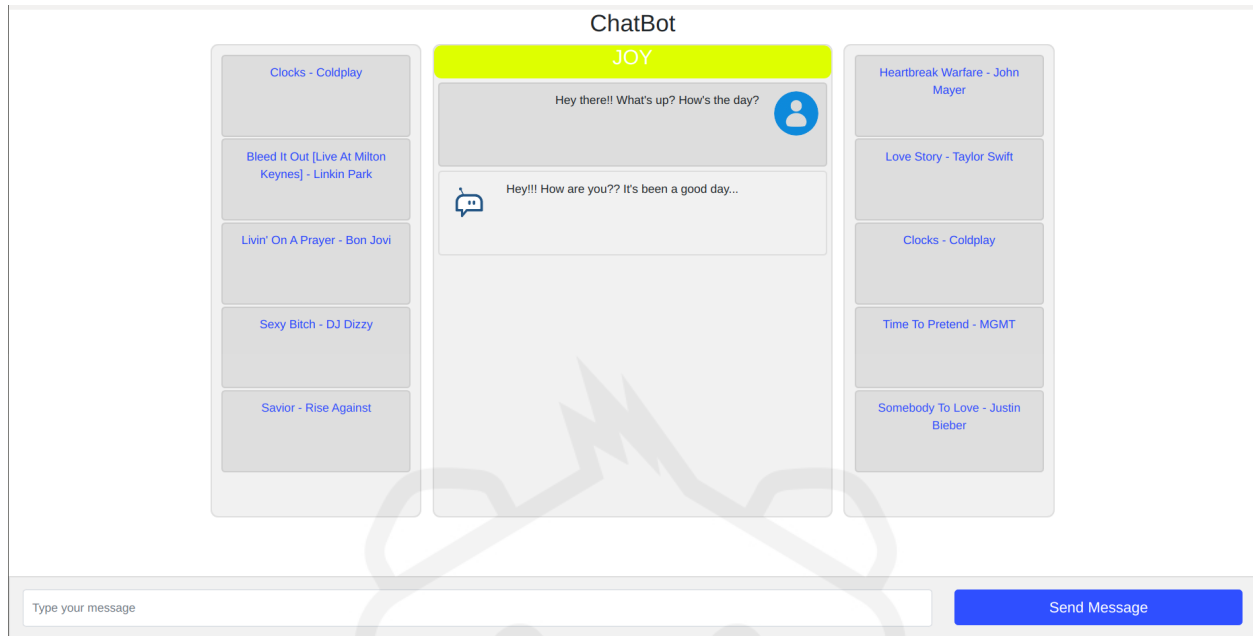
### Putting everything together and build the chatbot

Once you have understood the complete flow, you can do whatever you want now. You can go forward with building a command-line interface that collects all these things and does them. Or you can build a separate Flask/Django server that can do the above steps as a part of an API or a Web Page.

As a part of this milestone, we would be continuing with building a Django/Flask server.

### Requirements

- Building the following API's
  - /tone
    - **input:**
      - the context required for the chatbot
    - **output:** the tone-analysis of the conversation
  - /response
    - **input:**
      - the context required for the chatbot
      - the tone from /tone
    - **output:** the chatbot response
  - /songs
    - **input:**
      - the tag for getting songs from last.fm API
    - **output:**
      - list of songs
  - /similarsongs
    - **input:**
      - the song name
      - the artist name
    - **output:**
      - list of songs
- We can make a template inside your Django/flask app which has a page similar to this:



Here, on the left, I have the songs recommended to the user, and on the right-hand side are the songs which were recommended based on one of the songs which I clicked on.

Further, the emotion of the chat is shown at the top

## Reference

- [Building Rest API using Flask](#)
- [Building Rest API using Django](#)
- [Flask vs Django](#)

## Tip

Context of the Cakechat Chatbot is: at max last 3 conversations.

## Note

You can do this milestone in your own style.

## Bring it On!

Further, you can learn about docker and build a docker-compose file that provides all the services in one place.

## Expected Outcome

By the end of this milestone, you would have a working chatbot system with a good UI that recommends songs to the user along with chatting.