Description

Your company wants to test its music fingerprinting algorithms by implementing a monitoring system which will detect and display the music broadcast on radio stations across the world.

The fingerprinting team is responsible for building an infrastructure to download radio streams and run their algorithms to automatically detect what is being played on a given radio station at a given time. Your job is to help them store their results in a database.

The following information needs to be stored:

- radio stations, identified by their (unique) names
- performers, also identified by their names (which you may assume is unique)
- songs, identified by their names and their performer
- plays, identified by a song, a radio station, start and end time (with a precision to the second)

The fingerprint team need to be able to do the following things:

- insert a new radio station
- insert a new artist
- insert a new song
- insert a new play
- get all the plays for a given song between two dates
- get all the songs played on a given channel between two dates

The chief of product walks into your meeting by mistake and, after hearing your conversation, decides to add the following requirement in order to impress potential customers:

- get a top 40 for a given week and a list of channels: a list of 40 songs ordered by the number of times they were broadcast. For each song, provide their performer, playcount and the position they had the previous week.

Deliverable

What you are asked to provide is a web service that will allow the fingerprint team to use your software. The web service will listen to a port, will answer to GET / POST requests and return JSON dictionaries (specs below). You might want to use the provided script to test your code.

You can implement this prototype using the database and programming language of your choice, as long as they are free and available on linux. So, if you want to use Emacs Lisp as a programming language and a text file as a database, this is fine; however, we ask you to motivate your decision. Also, install notes and instructions on how to run your code would be nice.

Together with the source code for your program, we also ask you to answer a few questions:

- Tell us about your design choices, and why you made them
- What you provided is a prototype and therefore not a production ready software. But, of course, the marketing team doesn't know about that and already sold the finished product to three different customers. What do you think needs to be improved in your software in order to be used in production?
- How do you think your program would behave if the fingerprint team starts going crazy, inserts ten million songs and starts monitoring two thousand channels?

Specifications

To makes things simpler, all strings are UTF-8 encoded and all dates are in UTC, in the ISO 8601 format.

We provide a test script to help you check that your code is running fine. To run this script, you will need python >= 2.7; you can run it with the --add-data argument the first time you use it in order to import some random performers, songs and plays. It will make calls to your web service and check that the returned values are the ones expected. If you think there's something wrong in the script, just mention it in your answers.

The results should be json dictionaries, following this pattern:

```
{result: {...}, code: 0, errors: [ERROR_DESC_1, ERROR_DESC_2,...]}
```

if "code" is 0, it means that the query was successfully processed. errors is a list of strings with potential error messages (optional, only present if the code is different from 0).

Here are the calls it performs:

add_channel

```
POST /add_channel, data={name: 'channel_name'}
```

add_performer

```
POST /add_performer, data={name: 'performer'}
```

add_song

```
POST /add_song, data={title: 'song_name',
                      performer: 'performer_name'}
```

In the last call, if the performer does not exist, insert it. In all the previous three cases, if the entity already exists, just ignore the call.

add_play

```
POST /add_play, data={title: 'song_name',
                      performer: performer_name',
                      start: '2014-10-21T18:41:00',
                      end: '2014-10-21T18:44:00',
                      channel: 'channel_name'}
```

get_song_plays

```
GET /get_song_plays,
data={title: 'song_name',
      performer: performer_name',
      start: '2014-10-21T00:00:00',
      end: '2014-10-28T00:00:00'}

Returns:
{result: [{channel: 'channel',
           start: '2014-01-10T01:00:00',
           end: '2014-01-10T01:03:00'],...],
           code: 0}
```

get_channel_plays

```
GET /get_channel_plays,
    data={start: '2014-10-21T00:00:00',
          end: '2014-10-28T00:00:00',
          channel: 'channel'}

Returns:
{result: [{performer: 'performer_name',
           title: 'title',
           start: '2014-10-21T00:00:00',
           end: '2014-10-21T00:03:00'},...],
           code: 0}
```

get_top

```
GET /get_top,
data={channels: ['channel_name'],
      start: '2014-10-21T00:00:00',
      limit: 40}

Returns:
{result: [{performer: 'performer',
          title: 'title',
          plays: plays,
          previous_plays: previous_plays,
          rank: rank,
          previous_rank: previous_rank], ...],
          code: 0}
```

If the song was not in the top n last week, previous rank is null and plays is 0.