Web Engineering
Songregator

Dan Plămădeală (s3436624)
Alexander Fyodorov (s3274349)

October 14, 2019

1 Introduction

The purpose of the project is to develop a Web application, the main purpose of which is to provide users with information related to songs and their affiliated artists. The data required is taken from the Million Song Dataset prepared by CORGIS Dataset Project.

Users should be provided with the following minimum functionality:

1. Obtain information about all artists in the dataset.
   Optional filtering:
   (a) By artist name
   (b) By genre
2. Obtain all information about a specific song (identified by unique ID).
3. Obtain a list of songs by a specific artist and/or in a specific year.
4. Obtain a list of songs by artists in a specific genre.
5. Obtain a list of artists ranked by their popularity, with the possibility to subset this order, e.g. the top 50 artists.
6. Obtain a list of songs ranked by popularity.
   Optional filtering:
   (a) Select top **n** artists
   (b) Select bottom **n** artists
   (c) Select all artists with **hotttness** index in a specific range
7. Obtain descriptive statistics (mean, median, standard deviation) for the popularity of the songs for a particular artist. Optional filtering:
   (a) By year

2 API Design

For every API endpoint, the following **header field** holds:

Path parameter	Value
Content-Type	`application\json` ‖ `text\csv`

The default representation of resources is JSON.

A parameter is optional if it is typeset italic, for example the *Content-Type* parameter described earlier.

Used response codes:
200 - **OK**; this means that the request was successfully fulfilled.
400 - **Bad Request**; this means that there is a mistake in the syntax of the request.
403 - **Forbidden**; this means that the given Authorization header is not a valid one.
404 - **Not Found**; this means that the given resource was not found.
405 - **Method Not Allowed**; using an unsupported method, for example POST where PUT is expected.

1. All the artists in the dataset.

   Objective:
   Get all artists available in the dataset. Optionally, the user may ask for the artists to be filtered by the name and/or by their genre.
   The user may also request the results to be ordered ranked by their popularity and limit the size of the result to a given size.
   Calling this endpoint with empty parameters will yield a response containing all the artists in the dataset.

   Endpoint:

   ```
   GET /artist?name={name}
              &genre={genre}
              &ordered={ordered}
              &subset={size}
   ```

   Query parameters:

Query parameter	Value
name <string>	The name of the artist.
genre <string>	The genre of the artist.
ordered <boolean>	Whether the results should be ordered.
size <integer>	The size of the response.

   Response format:

   ```
   {
       "artists": [
           {
               "familiarity": float,
               "hotttnesss": float,
               "id": string,
               "latitude": float,
               "location": integer,
               "longitude": float,
               "name": string,
               "similar": float,
               "terms": string,
               "terms_freq": float
           },
           ...
   ```

```
    ]
}
```

2. Information about a song.

   Objective:
   Get all the available information about a song given a unique ID.

   Endpoint:

   `GET /song/{song ID}`

   Path parameters:

Path parameter	Value
song ID <string>	The unique ID of the song.

   Response:

```
{
    "artist_mbtags": float,
    "artist_mbtags_count": float,
    "bars_confidence": float,
    "bars_start": float,
    "beats_confidence": float,
    "beats_start": float,
    "duration": float,
    "end_of_fade_in": float,
    "hotttnesss": float,
    "id": string,
    "key": float,
    "key_confidence": float,
    "loudness": float,
    "mode": integer,
    "mode_confidence": float,
    "start_of_fade_out": float,
    "tatums_confidence": float,
    "tatums_start": float,
    "tempo": float,
    "time_signature": float,
    "time_signature_confidence": float,
    "title": integer,
    "year": integer
}
```

3. All songs in the dataset.

   Objective:
   Get all the songs in the dataset with optional parameters: artist, year of release and genre of the song.
   The user may also request the results to be ordered ranked by their popularity and limit the size of the result to a given size.
   Calling this endpoint with empty parameters will yield a response containing all the songs

in the dataset.

Endpoint:

```
GET /song?artist={artist}
         &year={year}
         &genre={genre}
         &ordered={ordered}
         &subset={size}
```

Query parameters:

Query parameter	Value
artist <string>	The artist of the song.
year <integer>	The year of release of the song.
genre <string>	The genre of the song.
ordered <boolean>	Whether the results should be ordered.
size <integer>	The size of the response.

Response:

```
{
    "songs": [
        {
            "artist_mbtags": float,
            "artist_mbtags_count": float,
            "bars_confidence": float,
            "bars_start": float,
            "beats_confidence": float,
            "beats_start": float,
            "duration": float,
            "end_of_fade_in": float,
            "hotttnesss": float,
            "id": string,
            "key": float,
            "key_confidence": float,
            "loudness": float,
            "mode": integer,
            "mode_confidence": float,
            "start_of_fade_out": float,
            "tatums_confidence": float,
            "tatums_start": float,
            "tempo": float,
            "time_signature": float,
            "time_signature_confidence": float,
            "title": integer,
            "year": integer
        },
        ...
    ]
}
```

4. Descriptive statistics of an artist.

Objective:
Descriptive statistics (mean, median, standard deviation) for the popularity of the songs for a particular artist with an optional filter by year.

Endpoint:

```
GET /popularity?artist={artist}&year={year}
```

Query parameters:

Query parameter	Value
artist <string>	Artist name.
year <integer>	Song release year.

Response:

```
{
    "mean": float,
    "median": float,
    "std": float
}
```

5. Delete all the songs of a given artist.

Objective:
Deletes all the entries (songs) in the dataset by the given artist.

Endpoint:

```
DELETE /song?artist={artist}
```

Query parameters:

Query parameter	Value
artist <string>	Artist name.

Response:
Returns an HTTP code which varies based on the input.

6. Update the location of the artist.

Objective:
Updates the location of the artist given the logitude and latitude coordinates.

Endpoint:

```
PUT /artist?artist={artist}
           &longitude={longitude}
           &latitude={latitude}
```

Query parameters:

Query parameter	Value
artist <string>	Artist name.
longitude <float>	Longitude coordinate.
latitude <float>	Latitude coordinate.

Response:
Returns an HTTP code which varies based on the input.

3 Architecture

Django is selected as the main framework. Reasons:

1. **It is flexible**. Since the app's functionality may be expanded, it is necessary to minimize any possible problems as much as possible.
2. **Database queries**. The database can be accessed right in the Python code, no SQL queries are used. It leads to an easier database access and an enhanced security (for example, by eliminating SQL injections).
3. **DRY**. By using Django, Web application and API backend are in the same codebase, making the development process easier and more reliable.
4. **Great support**. Django has a good documentation and supportive community.

SQLite was selected as a database management systems due to its compactness, portability and simplicity.

All the interaction with the backend is happening with a help of Django REST framework - a powerful and flexible toolkit that provides an extensive functionality out-of-the-box. It provides a great Serialization feature that translates complex datasets (e.g. queryset) to native Python datatypes, making it easy to render the data into JSON and XML. It is capable to generate a highly customizable view, providing a rich CRUD API to manipulate data with a minimum code. It allows to parse HTTP request parameters in a fast and convenient way. A demonstration screenshot can be found in the end of the document.

Planned features

1. Display the requested information in the browser window in a human-readable and fancy-looking way.
2. Provide statistics about the dataset (e.g.number of songs per artist)

API

So far, only the API for retrieving information about artists is implemented:

```
GET /artist?name={name}
           &genre={genre}
           &ordered={ordered}
           &subset={size}
```

where **name** and **genre** are filtering parameters, **ordered** states whether artists should be ordered by popularity (accepted values: **[1, True]**), and **subset** states the number of records to be displayed (only if **ordered=1/True**).

The view used for this purpose is **ModelViewSet**:

```
class ArtistViewSet(viewsets.ModelViewSet):
    """
    Generates a view for retrieving information about artists.
    """
    queryset = Song.objects.all()
    serializer_class = ArtistSerializer

    def get_queryset(self):
        queryset = self.queryset

        # If name parameter is specified
        name = self.request.query_params.get('name')
        if name:
            queryset = queryset.filter(artist_name=name)

        # If genre paratemer is specified
        genre = self.request.query_params.get('genre')
        if genre:
            queryset = queryset.filter(artist_terms=genre)

        # If ordered paratemer is specified
        ordered = self.request.query_params.get('ordered')
        if ordered in ['1', 'true']:
            queryset = queryset.order_by('-artist_hotttnesss')

            # If subset parameter is specified; applicable only if ordered is
    present.
            subset = self.request.query_params.get('subset')
            if subset and subset.isdigit():
                queryset = queryset[:int(subset)]

        return queryset
```

ArtistSerializer is a class that extracts only artists-related information:

```
class ArtistSerializer(serializers.HyperlinkedModelSerializer):
    """
    Extracts information about artists.
    """
    class Meta:
        model = Song
        fields = ('artist_familiarity',
                  'artist_hotttnesss',
                  'artist_id',
                  'artist_latitude',
                  'artist_location',
                  'artist_longitude',
                  'artist_name',
                  'artist_similar',
                  'artist_terms',
                  'artist_terms_freq')
```

Song is a model that represents the database:

```
class Song(models.Model):
    artist_familiarity = models.FloatField()
    artist_hotttnesss = models.FloatField()
    artist_id = models.CharField(max_length=80)
    artist_latitude = models.FloatField()
    artist_location = models.IntegerField()
    artist_longitude = models.FloatField()
    artist_name = models.CharField(max_length=80)
```

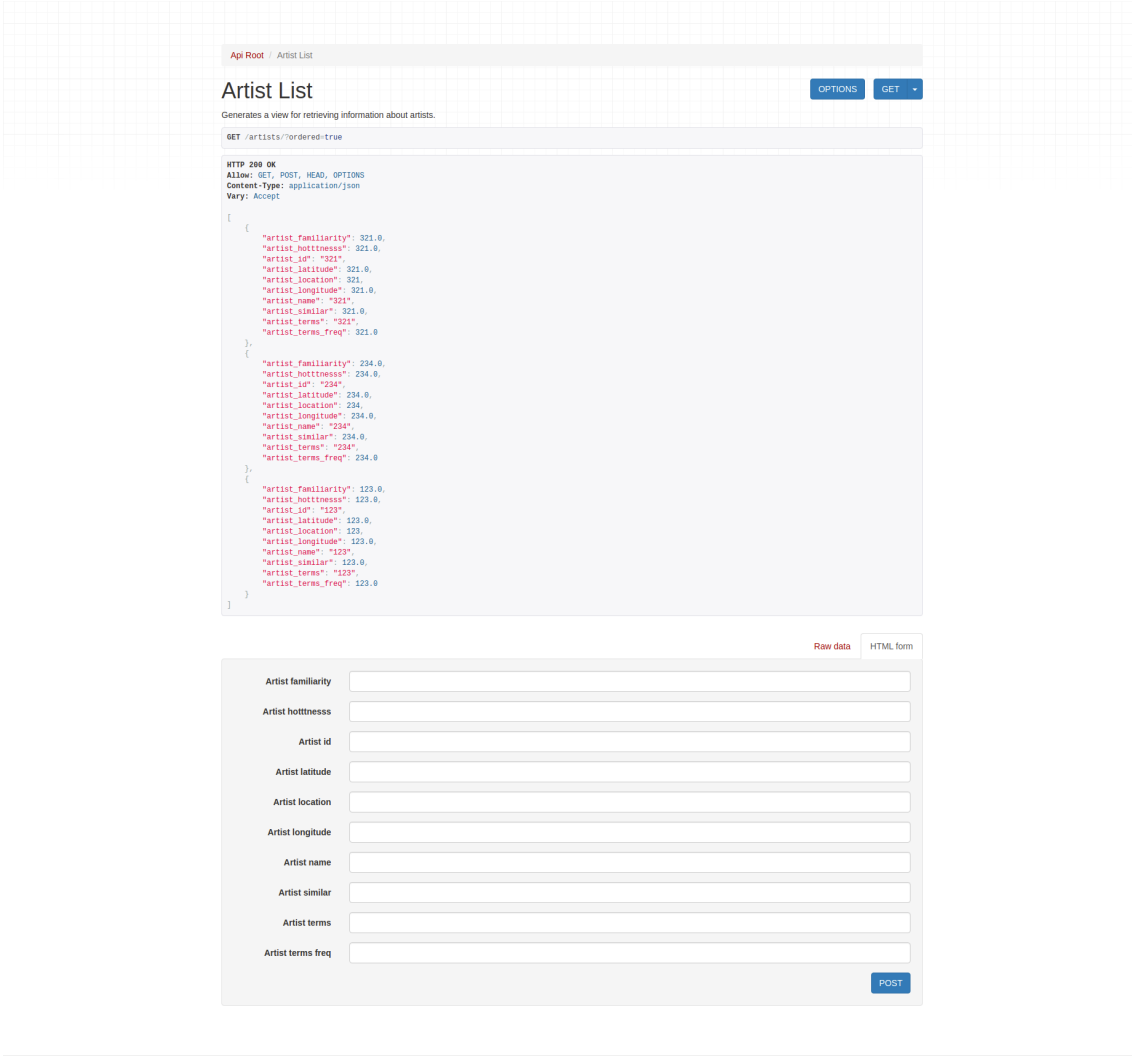
```
    artist_similar = models.FloatField()
    artist_terms = models.CharField(max_length=80)
    artist_terms_freq = models.FloatField()

    release_id = models.IntegerField()
    release_name = models.IntegerField()

    song_artist_mbtags = models.FloatField()
    song_artist_mbtags_count = models.FloatField()
    song_bars_confidence = models.FloatField()
    song_bars_start = models.FloatField()
    song_beats_confidence = models.FloatField()
    song_beats_start = models.FloatField()
    song_duration = models.FloatField()
    song_end_of_fade_in = models.FloatField()
    song_hotttnesss = models.FloatField()
    song_id = models.CharField(max_length=80)
    song_key = models.FloatField()
    song_key_confidence = models.FloatField()
    song_loudness = models.FloatField()
    song_mode = models.IntegerField()
    song_mode_confidence = models.FloatField()
    song_start_of_fade_out = models.FloatField()
    song_tatums_confidence = models.FloatField()
    song_tatums_start = models.FloatField()
    song_tempo = models.FloatField()
    song_time_signature = models.FloatField()
    song_time_signature_confidence = models.FloatField()
    song_title = models.IntegerField()
    song_year = models.IntegerField()

    def __str__(self):
        return str(self.song_title)
```

4 Conclusion



Work in Progress.