Web Engineering Milestone M1

Robert Patrona (S3322211) and Yona Moreda (S3324338)

Group 20

28 September 2019

API Design

Define data model

The data model consists of several songs and their corresponding attributes such as the year of the song, the hotness index which rates the popularity of the songs, duration etc. In addition, the data model also contains artists which consists of their id, name, location, familiarity, etc.

Split data into resources

The data is split into two major resources: the artists and the songs.

Name resources with URIs

Resource: The artists **URL**: `/artists` **URN**: `artist-name`
Resource: The songs **URL**: `/songs` **URN**: `song-id`

Design representations

The representation simply consists of the resource represented into its corresponding `json` or `csv` format. The `json` format will enable a representation of name/value pairs for the requested resources while `csv` provides the data with comma separeted values.

Link between resources

The resources are organized into "artists" and "songs" collections. Also, the artists and songs are linked with each other. For instance, they can have the same genre and artist name.

Model typical sequence of events

When querying a request, the user goes through a few steps. The first step is specifying the type of request, followed by the collection name, and then the query parameters. For instance, in the query `GET /songs?id=<song-id>`, `GET` denotes the type of request, `/songs` refers to the collection of songs, and `?id=<song-id>` is the query parameter. In addition, when doing a PATCH query, like in query 6, the user also needs to provide his own data, such as `{ song.title : <song-title> }`, which the API will use to update the entry in the dataset.

Define error conditions

A likely error condition might be requesting a PATCH query to update a song entry in the dataset. The error might occur if the provided song ID does correspond to any song.ID in the dataset. Another error condition might be requesting GET queries while providing an artist name which is not found in the dataset. In these cases, the API will respond with an error message.

Query Design

The following headers will be used in all requests.

`Content-Type: application/json` or `text/csv;`
`Content-Length: <length>`
`Last-Modified: <day-name>, <day> <month> <year> <hour>:<minute> GMT`
`Cache-Control: max-age=300`

For the cache control, we decided to use to use data up to 5 minutes of age. This is based on the assumption that the relevant data will not change within 5 minutes intervals.

1. Get all artists from the dataset, optionally filtered by artist name or music genre.
`GET /artists?name=<artist-name>&terms=<artist-terms>`

2. Get all information about a song from the database, identified by its unique song ID.
`GET /songs?id=<song-id>`

3. Get all songs by an artist given the artist name, or all songs in a specific year.
`GET /songs?artist=<artist-name>&year=<song-year>`

4. Get all songs by artists in a specific genre.
`GET /songs?terms=<artist-terms>`

5. Get the mean, median and standard deviation statistics of the popularity of the songs for a specific artist, optionally filtered by year.
`GET /songs/stats?name=<artist-name>&year=<song-year>`

6. Update the song title of a song in the dataset.
`PATCH /songs/<song-id> { song.title : <song-title> }`

The first 5 queries support the Read CRUD method, since they all retrieve information from to the client, without modifying it. The 6th query supports the Update CUD method, since it allows the user to modify a resource, namely the song title field of a song entry in the data set.

We will have 3 endpoints. One endpoint deals with the `songs` collection and supports the GET and PATCH queries and by extension the CRUD methods Read and Update. The other two endpoints deal with `artists` and `stats` collection, and they both support GET queries, or Read methods.