

**Project Web:**

**Deliverable 2**

Armand Ciutat Camps - 48056684R

Roger Castellví Rubinat - 48056998Q

Joel Aumedes Serrano - 48051307Y

Joel Farré Cortes -

Moises Bernaus Lechosa -

Marc Cervera Rosell

Delivery date: 21-05-2021

**Course 2020-21**

**Index**

# **1. Introduction**

In this 2nd Deliverable we implemented our web’s basic functionalities and we introduced ourselves into the world of automated web testing using Webdrivers.

## **Main changes**

* In Project Proposal, our application idea used Spotify’s API in order to obtain the musical likings of the users for generating data. Although, when starting to implement the application, it was not possible to accomplish authentication to Spotify’s API, so we changed the idea to a web where the user can save songs, artists or genres by typing them.
* In “models.py” we modified the models that we used in our app. Both favourite artist and favourite genre have the same attributes: *user*, *name*, *rating*, *description*.
* The favourite song has the same attributes and the name of the artist and the genre.
* In our app folder and our templates folder we organized the project using separated folders for artists, songs and genres to keep the code small, clean and structured.

# **2. Design and code explanations**

**Creation of instances**

For the creation of instances, it has been mandatory to create a class named *view*, i.e we will use a class together with his template in order to generate our website, instead of defining a function. This class, which is a subclass of CreateView allows us to create a form with the artist data. This form is created from a class we defined in *forms.py*, in which we defined said form to ask the user all the data in Favourite\_Artist model except the user, since we do not want a user to create artists to other users. With the class from the view, we defined that the user sent in the form is the same user that sent the request. Once the form is completed and the object has been created, the user is shown a page with the object details created with the django class *DetailView*, along with links to modify or delete the instance or going back to the list.

Song, genres and artists use the same procedure.

**Modification of Instances**

Once the instance has been created from its detail page, the user can enter another page to modify said instance. Using a process similar to the creation, but this time with UpdateView, the user now has permission to modify the rating and the description of the object, but not its name, or, in the case of the songs, neither the artist’s name. Once done or cancelled, the user goes back to the details page.

**Elimination of Instances**

For this case, we use a function in views file that receives the object’s id as an argument: This function basically deletes the object from the database and redirects the user back to the list. There can not be problems with the type of object to delete, because the URL we use to delete an object has the type of object in it.

**Tests**

Au mems

**API Use**

Our website used jQuery library in order to offer the user the autocomplete function in certain fields when creating an object to the website. The function can receive an argument *source*, i.e, from which will obtain the registries that will offer to the user when he types 3 different type values: a JavaScript list, a String or a function. JavaScript list does not allow us to be flexible enough and the function is too advanced for our knowledge of JavaScript. For that reason, we opted for String library. This String has to be a URL, so that if the URL is: http://www.exemple.com, this domain has to answer a JSON answer with the showing results when it is accessed using http://www.exemple.com?term=thing-to-search. If we pass a URL to our website to *source* argument, and we define a view that obtains the data and returns it with a JsonResponse, we can do the request to the API in a much easier way.

When the user wants to create an artist, the name attribute is the one that has autocomplete, doing a request to a free API with musical data called MusicBrainz. This autocomplete does not work very well, since MusicBrainz only has one method for searching and does not search starting from the beginning. So if the user types “Que” waiting for “Queen” to pop up, before it will get of bands with names like Que, [.que], or Gee-Que since they are similar to the search term.

When the user wants to create a genre, the name attribute does the autocomplete, reading from a file which contains approximately 1800 different genres.

When the user wants to create a song, autocomplete is made in 2 attributes: genre and artist name. Genre happens to be the same as if a genre is created, reading from the file. On the other hand, artist name, instead of doing a search to the MusicBrainz API, gives priority to the artists that the artist has added, so that if user has added any artist, it will pop up early in autocomplete.