|  |  |
| --- | --- |
| Resumen |  |
| Número total de pruebas realizadas | 56 |
| Número de pruebas automatizadas | 51 (91%) |

TestMusixMatchResource

|  |  |
| --- | --- |
| ID | **Test 8 - testSearchNull()** |
| Description | Test aimed to detect errors on the implementation of searches in MusixMatch using RESTful services. (I) |
| Input | We use the *MusixMatchResource* class in order to invoke the service using the URI *http://api.musixmatch.com/ws/1.1/matcher.lyrics.get ?apikey=&q\_track=&q\_artist=* from our application.  Where:   * apikey = our private API key supplied by MusixMatch * q\_track = null * q\_artist = null |
| Expected Output | As there are some require fields that need to be parsed before the REST request, the application will raise a NullPointerException when trying to parse the null values. |
| Result | **SUCCESS** |
| Automatized | Yes |

|  |  |
| --- | --- |
| ID | **Test 9 - testSearchTrackNull()** |
| Description | Test aimed to detect errors on the implementation of searches in MusixMatch using RESTful services. (II) |
| Input | We use the *MusixMatchResource* class in order to invoke the service using the URI *http://api.musixmatch.com/ws/1.1/matcher.lyrics.get ?apikey=&q\_track=&q\_artist=* from our application.  Where:   * apikey = our private API key supplied by MusixMatch * q\_track = null * q\_artist = “” |
| Expected Output | As there are some require fields that need to be parsed before the REST request, the application will raise a NullPointerException when trying to parse the null values. |
| Result | **SUCCESS** |
| Automatized | Yes |

|  |  |
| --- | --- |
| ID | **Test 10 - testSearchArtistNull()** |
| Description | Test aimed to detect errors on the implementation of searches in MusixMatch using RESTful services. (III) |
| Input | We use the *MusixMatchResource* class in order to invoke the service using the URI *http://api.musixmatch.com/ws/1.1/matcher.lyrics.get ?apikey=&q\_track=&q\_artist=* from our application.  Where:   * apikey = our private API key supplied by MusixMatch * q\_track = “” * q\_artist = null |
| Expected Output | As there are some require fields that need to be parsed before the REST request, the application will raise a NullPointerException when trying to parse the null values. |
| Result | **SUCCESS** |
| Automatized | Yes |

|  |  |
| --- | --- |
| ID | **Test 11 - testSearchEmpty()** |
| Description | Test aimed to detect errors on the implementation of searches in MusixMatch using RESTful services. (IV) |
| Input | We use the *MusixMatchResource* class in order to invoke the service using the URI *http://api.musixmatch.com/ws/1.1/matcher.lyrics.get ?apikey=&q\_track=&q\_artist=* from our application.  Where:   * apikey = our private API key supplied by MusixMatch * q\_track = “” * q\_artist = “” |
| Expected Output | The fields now indicate invalid track and artist values in the MusixMatch server. We will verify that the lyrics on the Lyrics class (Java class resulting from a mapping of the returned JSON data) match the String “*Lyrics aren't available for this track”.* |
| Result | **SUCCESS** |
| Automatized | Yes |

|  |  |
| --- | --- |
| ID | **Test 12 - testSearchPageFound()** |
| Description | Test aimed to detect errors on the implementation of searches in MusixMatch using RESTful services. (V) |
| Input | We use the *MusixMatchResource* class in order to invoke the service using the URI *http://api.musixmatch.com/ws/1.1/matcher.lyrics.get ?apikey=&q\_track=&q\_artist=* from our application.  Where:   * apikey = our private API key supplied by MusixMatch * q\_track = “Occidentali's Karma” * q\_artist = “Francesco Gabanni” |
| Expected Output | The fields now indicate valid track and artist values in the MusixMatch server. We will verify that the lyrics on the Lyrics class (Java class resulting from a mapping of the returned JSON data) don’t match the String “*Lyrics aren't available for this track”.* Furthermore*,* they need to contain the sentence “Essere o dover essere”, which is a sentence that is contained in Francesco Gabanni’s song. |
| Result | **SUCCESS** |
| Automatized | Yes |

TestSongkickResource

|  |  |
| --- | --- |
| ID | **Test 13 - testSearchNull()** |
| Description | Test aimed to detect errors on the implementation of searches in SongKick using RESTful services. (I) |
| Input | We use the *SongKickResource* class in order to invoke the service using the URI *http://api.songkick.com/api/3.0/events.json?*  *apikey=&artist\_name=* from our application.  Where:   * apikey = our private API key supplied by MusixMatch * artist\_name = null |
| Expected Output | As there are some require fields that need to be parsed before the REST request, the application will raise a NullPointerException when trying to parse the null values. |
| Result | **SUCCESS** |
| Automatized | Yes |

|  |  |
| --- | --- |
| ID | **Test 14 - testSearchEmpty()** |
| Description | Test aimed to detect errors on the implementation of searches in SongKick using RESTful services. (II) |
| Input | We use the *SongKickResource* class in order to invoke the service using the URI *http://api.songkick.com/api/3.0/events.json?*  *apikey=&artist\_name=* from our application.  Where:   * apikey = our private API key supplied by SongKick * artist\_name = “” |
| Expected Output | The fields now indicate invalid artist\_name value in the SongKick server. We will verify that the status property on the ConcertsSearch class (Java class resulting from a mapping of the returned JSON data) is ‘ok’ as the server should have received our request. Also, we will check that the total entries and the results are 0 and null respectively. |
| Result | **SUCCESS** |
| Automatized | Yes |

|  |  |
| --- | --- |
| ID | **Test 15 - testSearchPageFound()** |
| Description | Test aimed to detect errors on the implementation of searches in SongKick using RESTful services. (III) |
| Input | We use the *SongKickResource* class in order to invoke the service using the URI *http://api.songkick.com/api/3.0/events.json?*  *apikey=&artist\_name=* from our application.  Where:   * apikey = our private API key supplied by SongKick * artist\_name = “Lady Gaga” |
| Expected Output | The fields now indicate valid artist\_name value in the SongKick server. We will verify that the status property on the ConcertsSearch class (Java class resulting from a mapping of the returned JSON data) is ‘ok’ as the server should have received our request. Also, we will check that the total entries are greater than 0 and the results not null. In addition, for each result we will verify that the status property equals ‘ok’. |
| Result | **SUCCESS** |
| Automatized | Yes |

TestSpotifyResource

|  |  |
| --- | --- |
| ID | **Test 16 - testAddTrackNull()** |
| Description | Test aimed to detect errors on the implementation of adding a resource in Spotify using RESTful services. (I) |
| Input | We use the Spotify*Resource* class in order to invoke the service using the URI https://api.spotify.com/v1/me/tracks?access\_token=&ids=  from our application.  Where:   * access\_token = personal access token of the actual user * ids = null |
| Expected Output | The ids field has an invalid value, so it will not add any song to the user’s profile. |
| Result | **SUCCESS** |
| Automatized | Yes |

|  |  |
| --- | --- |
| ID | **Test 17 - testAddNonExistingTrack()** |
| Description | Test aimed to detect errors on the implementation of adding a resource in Spotify using RESTful services. (II) |
| Input | We use the Spotify*Resource* class in order to invoke the service using the URI https://api.spotify.com/v1/me/tracks?access\_token=&ids=  from our application.  Where:   * access\_token = personal access token of the actual user * ids = “blanche” |
| Expected Output | The ids field has an invalid value, so it will not add any song to the user’s profile. |
| Result | **SUCCESS** |
| Automatized | Yes |

|  |  |
| --- | --- |
| ID | **Test 18 - testAddNonExistingTrack()** |
| Description | Test aimed to detect errors on the implementation of adding a resource in Spotify using RESTful services. (III) |
| Input | We use the Spotify*Resource* class in order to invoke the service using the URI https://api.spotify.com/v1/me/tracks?access\_token=&ids=  from our application.  Where:   * access\_token = personal access token of the actual user * ids = “2fQxE0jVrjNMT9oJAXtSJR” |
| Expected Output | The ids field has a valid value this time, but now it matches a song that is currently in the user’s library, so we expect true as result. |
| Result | **SUCCESS** |
| Automatized | Yes |

|  |  |
| --- | --- |
| ID | **Test 19 - testAddNonExistingTrack()** |
| Description | Test aimed to detect errors on the implementation of adding a resource in Spotify using RESTful services. (IV) |
| Input | We use the Spotify*Resource* class in order to invoke the service using the URI https://api.spotify.com/v1/me/tracks?access\_token=&ids=  from our application.  Where:   * access\_token = personal access token of the actual user * ids = “3yvZ1VUso9Xi7EvIyLaj4o” |
| Expected Output | The ids field has a valid value this time, so we expect true as result. |
| Result | **SUCCESS** |
| Automatized | Yes |

|  |  |
| --- | --- |
| ID | **Test 20 - testAddAlbumNull()** |
| Description | Test aimed to detect errors on the implementation of adding a resource in Spotify using RESTful services. (V) |
| Input | We use the Spotify*Resource* class in order to invoke the service using the URI https://api.spotify.com/v1/me/albums?access\_token=&ids=  from our application.  Where:   * access\_token = personal access token of the actual user * ids = null |
| Expected Output | The ids field has an invalid value, so it will not add any album to the user’s profile. |
| Result | **SUCCESS** |
| Automatized | Yes |

|  |  |
| --- | --- |
| ID | **Test 21 - testAddNonExistingAlbum()** |
| Description | Test aimed to detect errors on the implementation of adding a resource in Spotify using RESTful services. (VI) |
| Input | We use the Spotify*Resource* class in order to invoke the service using the URI https://api.spotify.com/v1/me/albums?access\_token=&ids=  from our application.  Where:   * access\_token = personal access token of the actual user * ids = “blanche” |
| Expected Output | The ids field has an invalid value, so it will not add any album to the user’s profile. |
| Result | **SUCCESS** |
| Automatized | Yes |

|  |  |
| --- | --- |
| ID | **Test 22 - testAddNonExistingTrack()** |
| Description | Test aimed to detect errors on the implementation of adding a resource in Spotify using RESTful services. (VII) |
| Input | We use the Spotify*Resource* class in order to invoke the service using the URI https://api.spotify.com/v1/me/albums?access\_token=&ids=  from our application.  Where:   * access\_token = personal access token of the actual user * ids = “5AMOKSM1ftb3opIbGT2d4q” |
| Expected Output | The ids field has a valid value this time, but now it matches a album that is currently in the user’s library, so we expect true as result. |
| Result | **SUCCESS** |
| Automatized | Yes |

|  |  |
| --- | --- |
| ID | **Test 23 - testAddNonExistingTrack()** |
| Description | Test aimed to detect errors on the implementation of adding a resource in Spotify using RESTful services. (VIII) |
| Input | We use the Spotify*Resource* class in order to invoke the service using the URI https://api.spotify.com/v1/me/albums?access\_token=&ids=  from our application.  Where:   * access\_token = personal access token of the actual user * ids = “3Ywlsvgu3H6L3q9NHydNR3” |
| Expected Output | The ids field has a valid value this time, so we expect true as result. |
| Result | **SUCCESS** |
| Automatized | Yes |

TestWikipediaResource

|  |  |
| --- | --- |
| ID | **Test 45 - testSearchNull()** |
| Description | Test aimed to detect errors on the implementation of searches in Wikipedia using RESTful services. (I) |
| Input | We use the *WikipediaResource* class in order to invoke the service using the URI *https://en.wikipedia.org/w/api.php?action=query&format=json&prop=extracts&exintro=1&formatversion=2&titles=* from our application.  Where:   * titles = null |
| Expected Output | As there are some require fields that need to be parsed before the REST request, the application will raise a NullPointerException when trying to parse the null values. |
| Result | **SUCCESS** |
| Automatized | Yes |

|  |  |
| --- | --- |
| ID | **Test 46 - testSearchEmpty()** |
| Description | Test aimed to detect errors on the implementation of searches in Wikipedia using RESTful services. (I) |
| Input | We use the *WikipediaResource* class in order to invoke the service using the URI *https://en.wikipedia.org/w/api.php?action=query*  *&format=json&prop=extracts&exintro=1&formatversion=2&titles=* from our application.  Where:   * titles = “” |
| Expected Output | The titles field have an invalid value. The returned BiographySearch (Java class resulting from a mapping of the returned JSON data) shouldn’t be null and its batchCompleted property should be true (it shows whether the Wikipedia received the request or not). Furthermore, the query property (the actual biography) should be null. |
| Result | **SUCCESS** |
| Automatized | Yes |

|  |  |
| --- | --- |
| ID | **Test 47 - testSearchPageNotFound()** |
| Description | Test aimed to detect errors on the implementation of searches in SongKick using RESTful services. (III) |
| Input | We use the *WikipediaResource* class in order to invoke the service using the URI *https://en.wikipedia.org/w/api.php?action=query*  *&format=json&prop=extracts&exintro=1&formatversion=2&titles=* from our application.  Where:   * titles = “Antonio Pagudo” |
| Expected Output | The titles field have valid values that is not present in the DB. The returned BiographySearch (Java class resulting from a mapping of the returned JSON data) shouldn’t be null and its batchCompleted property should be true (it shows whether the Wikipedia received the request or not). Furthermore, the missing property should be true there is no article related to that query. |
| Result | **SUCCESS** |
| Automatized | Yes |

|  |  |
| --- | --- |
| ID | **Test 48 - testSearchPageFound()** |
| Description | Test aimed to detect errors on the implementation of searches in SongKick using RESTful services. (III) |
| Input | We use the *WikipediaResource* class in order to invoke the service using the URI *https://en.wikipedia.org/w/api.php?action=query*  *&format=json&prop=extracts&exintro=1&formatversion=2&titles=* from our application.  Where:   * titles = “Lady Gaga” |
| Expected Output | The titles field have valid values that is present in the DB. The returned BiographySearch (Java class resulting from a mapping of the returned JSON data) shouldn’t be null and its batchCompleted property should be true (it shows whether the Wikipedia received the request or not). Furthermore, the query (the actual biography), pageId and extract properties shouldn’t be null, nevertheless missing should be null. |
| Result | **SUCCESS** |
| Automatized | Yes |

TestYoutubeResource

|  |  |
| --- | --- |
| ID | **Test 49 - testSearchNull()** |
| Description | Test aimed to detect errors on the implementation of searches in Youtube using RESTful services. (I) |
| Input | We use the *YoutubeResource* class in order to invoke the service using the URI *https://www.googleapis.com/youtube/v3/search?part=id&q=*  *&type=video&key=* from our application.  Where:   * key = our private API key supplied by Youtube * q = null |
| Expected Output | As there are some require fields that need to be parsed before the REST request, the application will raise a NullPointerException when trying to parse the null values. |
| Result | **SUCCESS** |
| Automatized | Yes |

|  |  |
| --- | --- |
| ID | **Test 50 - testSearchNotFound()** |
| Description | Test aimed to detect errors on the implementation of searches in Youtube using RESTful services. (II) |
| Input | We use the *YoutubeResource* class in order to invoke the service using the URI *https://www.googleapis.com/youtube/v3/search?part=id&q=*  *&type=video&key=* from our application.  Where:   * key = our private API key supplied by Youtube * q = “Rodolfo Chikilcuatre Baila el Chiki Chiki (feat. Tata Golosa) - Original Mix” |
| Expected Output | The q field have valid values that is not present in the DB. The returned VideosSearch (Java class resulting from a mapping of the returned JSON data) shouldn’t be null and neither does its pageInfo property. Furthermore, the totalResults should be 0 hence the list of items should be empty. |
| Result | **SUCCESS** |
| Automatized | Yes |

|  |  |
| --- | --- |
| ID | **Test 51 - testSearchFound()** |
| Description | Test aimed to detect errors on the implementation of searches in Youtube using RESTful services. (II) |
| Input | We use the *YoutubeResource* class in order to invoke the service using the URI *https://www.googleapis.com/youtube/v3/search?part=id&q=*  *&type=video&key=* from our application.  Where:   * key = our private API key supplied by Youtube * q = “Dami Im Sound Of Silence” |
| Expected Output | The q field have valid values that is not present in the DB. The returned VideosSearch (Java class resulting from a mapping of the returned JSON data) shouldn’t be null and neither does its pageInfo property. Furthermore, the totalResults should be greater than 0 hence the list of items shouldn’t be empty. |
| Result | **SUCCESS** |
| Automatized | Yes |