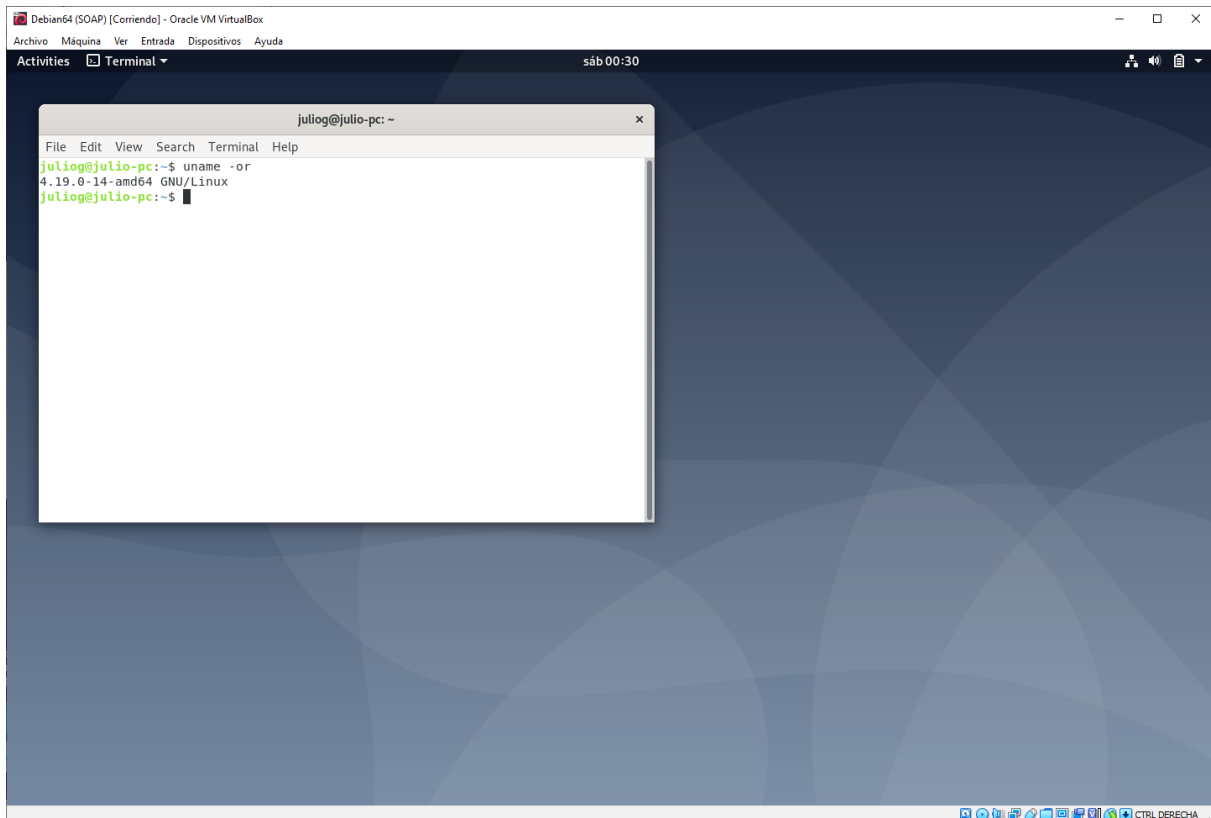
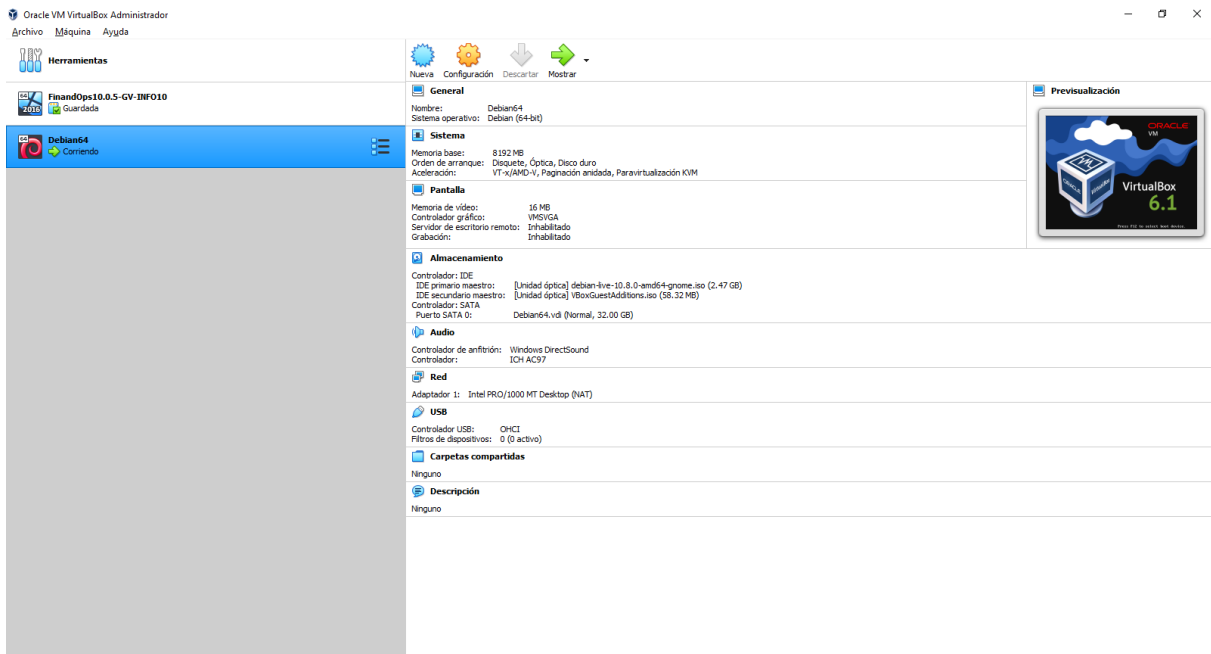
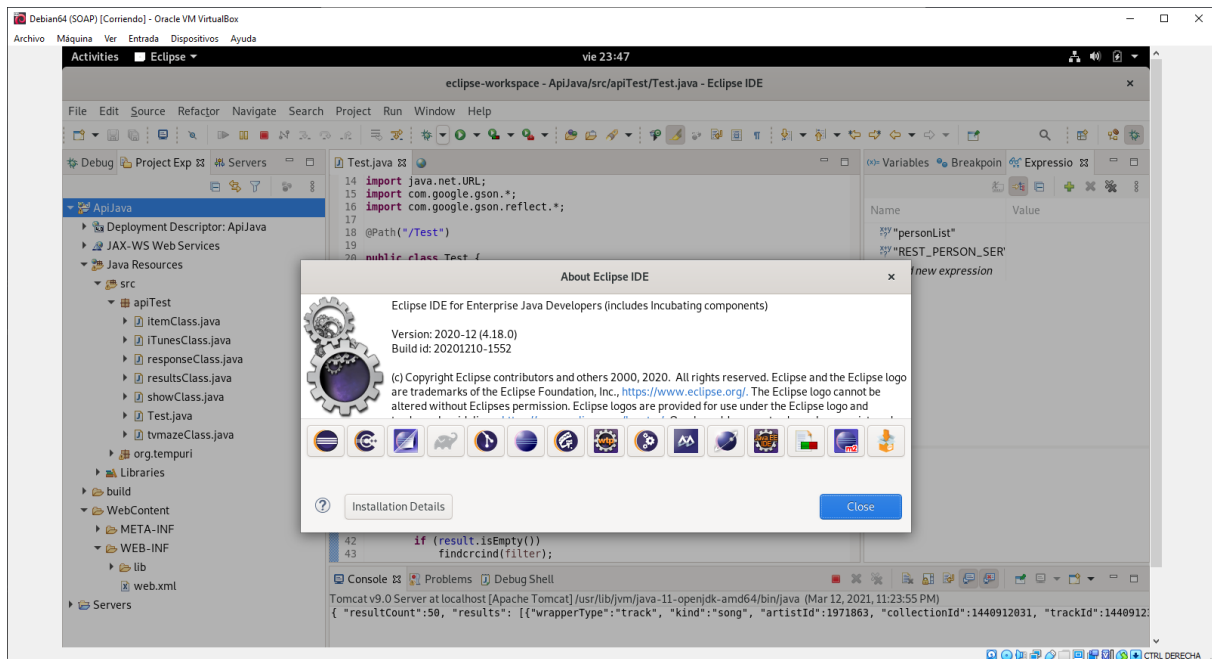


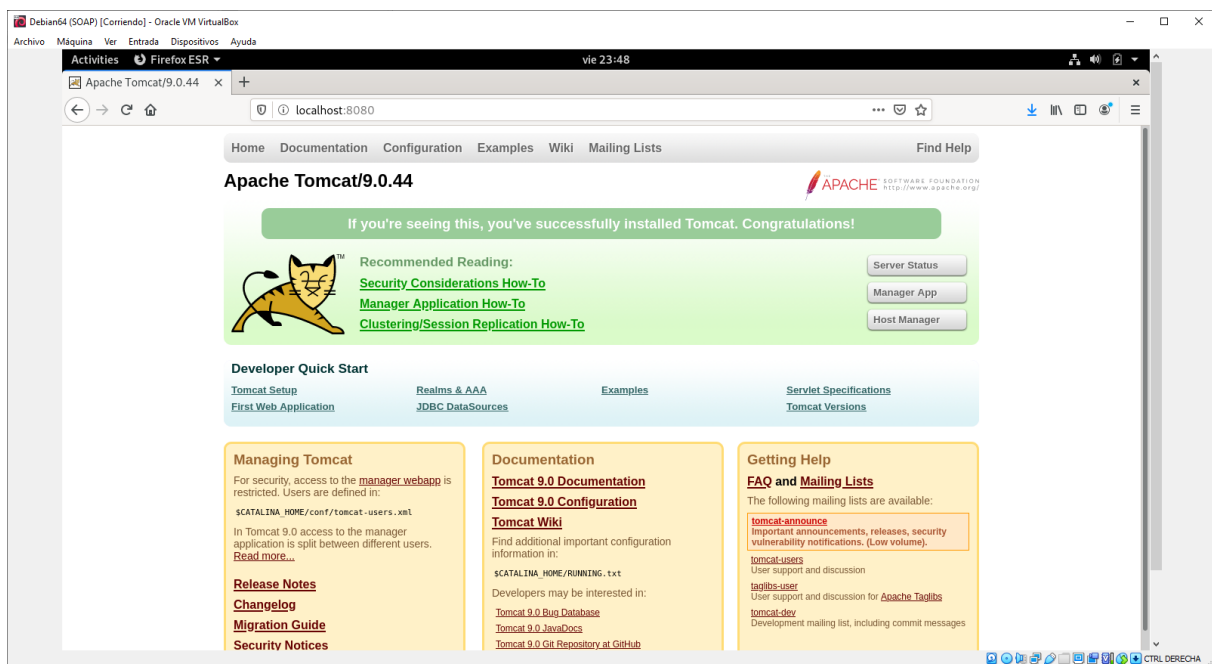
Se utilizó un Linux Debian 10 limpio en una máquina virtual.



Como IDE se utilizó Eclipse



Como servidor de aplicaciones se utilizó Apache Tomcat v9.0.44



Se validó la respuesta de cada una de los 3 orígenes o fuentes de datos.

## API-TEST

<http://api.tvmaze.com/search/shows?q=girls> (REST)

Postman interface showing a GET request to `http://api.tvmaze.com/search/shows?q=girls`. The response is a JSON object:

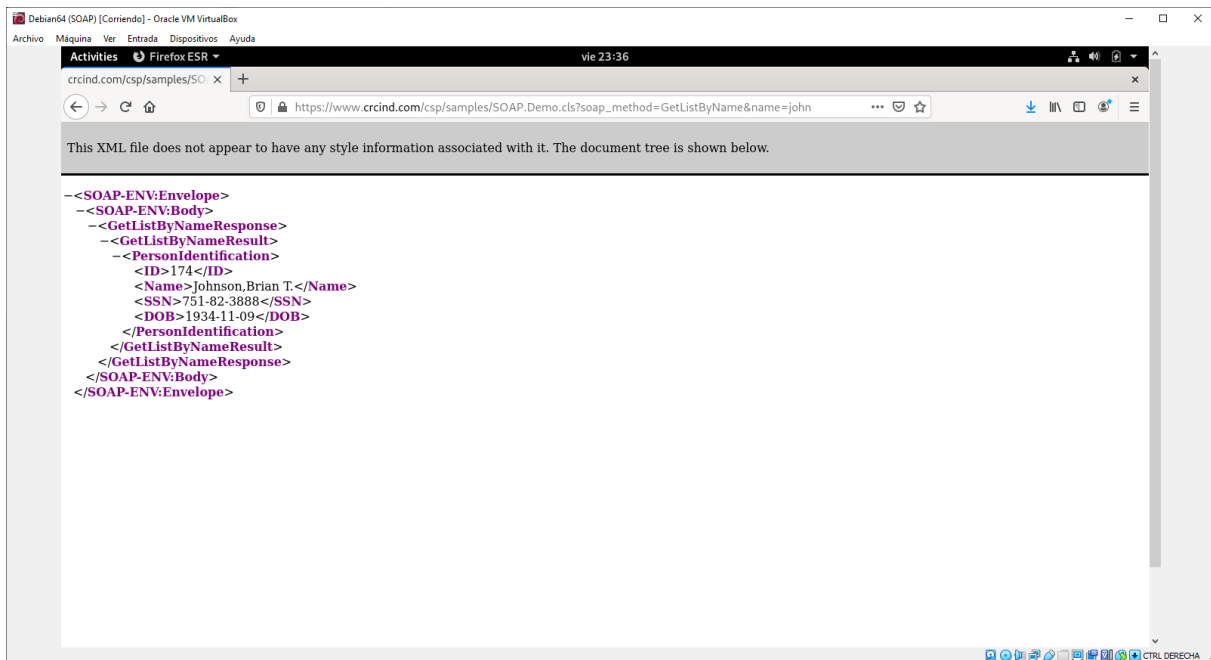
```
{
  "score": 17.776824,
  "show": {
    "id": 139,
    "url": "https://www.tvmaze.com/shows/139/girls",
    "name": "Girls",
    "type": "Scripted",
    "language": "English",
    "genres": [
      "Drama",
      "Romance"
    ],
    "status": "Ended",
    "runtime": 30,
    "premiered": "2012-04-10",
    "officialSite": "http://www.hbo.com/girls",
    "schedule": {
      "time": "22:00",
      "days": [
        "Sunday"
      ]
    },
    "rating": {
      "average": 6.6
    },
    "weight": 91,
    "network": {
      "id": 0,
      "name": "HBO",
      "country": {
        "name": "United States",
        "code": "US",
        "timezone": "America/New_York"
      }
    },
    "externals": {
      ...
    }
  }
}
```

<https://itunes.apple.com/search?term=girls> (REST)

Postman interface showing a GET request to `https://itunes.apple.com/search?term=girls`. The response is a JSON object:

```
{
  "resultCount": 100,
  "results": [
    {
      "wrapperType": "track", "kind": "song", "artistId": 1971863, "collectionId": 1448912831,
      "trackId": 1448912830, "artistName": "Beastie Boys", "collectionName": "Licensed to Ill",
      "trackName": "Girls", "collectionCensoredName": "Licensed to Ill",
      "trackCensoredName": "Girls", "artistViewUrl": "https://music.apple.com/us/artist/beastie-boys/1971863?uo=ad", "collectionViewUrl": "https://music.apple.com/us/album/girls/1448912831?uo=ad", "trackViewUrl": "https://music.apple.com/us/album/girls/1448912831?uo=ad",
      "previewUrl": "https://audio-ssl.itunes.apple.com/itunes-assets/AudioPreview125/v4/61/9f/619fa2-767a-7838-8768-0141c4e88dc/ncf149183889787478490.plus.sac.p.m4a",
      "artworkUrl100": "https://is1-ssl.mzstatic.com/image/thumb/Music124/v4/69/66/34/6966343d-8f4c-b09a-ae1b-82c0b0f77640/source/200x200.jpg", "artworkUrl300": "https://is1-ssl.mzstatic.com/image/thumb/Music124/v4/69/66/34/6966343d-8f4c-b09a-ae1b-82c0b0f77640/source/300x300.jpg", "artworkUrl600": "https://is1-ssl.mzstatic.com/image/thumb/Music124/v4/69/66/34/6966343d-8f4c-b09a-ae1b-82c0b0f77640/source/600x600.jpg", "collectionPrice": 9.99,
      "trackPrice": 1.29, "releaseDate": "1986-11-18T12:00:00Z",
      "collectionExplicitness": "notExplicit", "trackExplicitness": "notExplicit",
      "discCount": 1, "discNumber": 1, "trackCount": 11, "trackNumber": 6,
      "trackTimeMillis": 132787, "country": "USA", "currency": "USD",
      "summaryGenreNames": "Hip-Hop/Rap", "isStreamable": true,
      "wrapperType": "track", "kind": "song", "artistId": 1081091, "collectionId": 284118362,
      "trackId": 284118361, "artistName": "Girls", "collectionName": "Annie (Original Studio Cast)", "trackName": "It's a Hard Knock Life", "collectionCensoredName": "Annie (Original Studio Cast)", "trackCensoredName": "It's a Hard Knock Life",
      "collectionArtistId": 1481379, "collectionArtistName": "Charles Strouse, Martin Charnin & National Symphony Orchestra, UK", "collectionViewUrl": "https://music.apple.com/us/artist/girls/284118361", "collectionViewUrl": "https://music.apple.com/us/album/its-a-hard-knock-life/284118361?uo=ad", "trackViewUrl": "https://music.apple.com/us/album/its-a-hard-knock-life/284118361?uo=ad",
      "previewUrl": "https://audio-ssl.itunes.apple.com/itunes-assets/AudioPreview125/v4/61/9f/619fa2-767a-7838-8768-0141c4e88dc/ncf149183889787478490.plus.sac.p.m4a"
    }
  ]
}
```

[http://www.crcind.com/csp/samples/SOAP.Demo.cls?soap\\_method=GetListByName&name=john](http://www.crcind.com/csp/samples/SOAP.Demo.cls?soap_method=GetListByName&name=john) (SOAP)



La clase principal es Test.java

En esta clase se agregaron las Url para cada servicio para consultar datos

```
25 private String REST_TVMAZE_SERVICE_URL = "http://api.tvmaze.com/search/shows?q=";
26 private String REST_ITUNES_SERVICE_URL = "https://itunes.apple.com/search?term=";
27 private String REST_PERSON_SERVICE_URL = "http://www.crcind.com/csp/samples/SOAP.Demo.cls";
```

Se creó un método GET con el nombre `findByName` que recibe como parámetro una cadena para hacer la búsqueda en las 3 fuentes de datos.

```
33 @GET
34 @Path("/find")
35 public responseClass findByName(@QueryParam("filter") String filter){
36     if (result.isEmpty())
37         findShows(filter);
38
39     if (result.isEmpty())
40         findiTunes(filter);
41
42     if (result.isEmpty())
43         findCrcind(filter);
44
45     if (result.isEmpty())
46         result = PASS;
47
48     response.setResult(result);
49     response.setList(list);
50
51     return response;
52 }
```

Cada búsqueda se realiza utilizando un método diferente (enviando el parámetro `filter`) ya que para buscar shows, canciones y películas el origen de datos es un servicio REST y para buscar personas es un método SOAP.

Para realizar la búsqueda de shows se usa el metodo `findShows`. Se usa la Url mencionada anteriormente, se indica que es una petición GET y que el tipo de contenido es json. Si el código de respuesta es diferente de 200 (OK) se termina esta

solicitud, de lo contrario se lee completamente la respuesta, la cual es una cadena json y se convierte a una lista de objetos de tipo tvmazeClass.

```

54 private void findShows(String filter) {
55     try {
56
57         URL url = new URL(REST_TVMAZE_SERVICE_URL + filter);
58         HttpURLConnection conn = (HttpURLConnection) url.openConnection();
59         conn.setRequestMethod("GET");
60         conn.setRequestProperty("Accept", "application/json");
61         if (conn.getResponseCode() != 200) {
62             throw new RuntimeException("Failed : HTTP Error code : "
63                                     + conn.getResponseCode());
64         }
65         InputStreamReader in = new InputStreamReader(conn.getInputStream());
66         BufferedReader br = new BufferedReader(in);
67         String output;
68         while ((output = br.readLine()) != null) {
69             System.out.println(output);
70             if (output.isEmpty() == false)
71             {
72                 Gson gson = new Gson();
73
74                 List<tvmazeClass> shows = gson.fromJson(output, new TypeToken<List<tvmazeClass>>(){}.getType());
75
76                 if (shows.isEmpty() == false) {
77                     for (tvmazeClass each: shows) {
78                         //result += each.getShow().getName() + "<br/>";
79
80                         list.add(new itemClass()
81                                 .withType("show")
82                                 .withName(each.getShow().getName())
83                                 .withUrl(each.getShow().getUrl())
84                                 .withSummary(each.getShow().getSummary())
85                                 );
86                     }
87                 }
88             }
89         }
90         conn.disconnect();
91     } catch (Exception e) {
92         System.out.println("Exception in NetClientGet:- " + e);
93         result = FAIL + " - " + e;
94     }
95 }

```

La clase tvmazeClass contiene un objeto de la clase showClass.

```

1 package apiTest;
2
3 public class tvmazeClass {
4     private float score;
5     private showClass show;
6
7     // Getter Methods
8
9     public float getScore() {
10         return score;
11     }
12
13     public showClass getShow() {
14         return show;
15     }
16
17     // Setter Methods
18
19     public void setScore( float score ) {
20         this.score = score;
21     }
22
23     public void setShow( showClass show ) {
24         this.show = show;
25     }
26 }

```

La clase showClass contiene la información que posteriormente se enviara en la respuesta de las coincidencias que cumple el criterio de búsqueda.

```
1 package apiTest;
2
3 public class showClass {
4     private float id;
5     private String url;
6     private String name;
7     private String type;
8     private String summary;
9
10    public float getId() {
11        return id;
12    }
13
14    public String getUrl() {
15        return url;
16    }
17
18    public String getName() {
19        return name;
20    }
21
22    public String getType() {
23        return type;
24    }
25
26    public String getSummary() {
27        return summary;
28    }
29 }
```

Para realizar la búsqueda de canciones y películas se usa el metodo `findiTunes`. Se usa la Url mencionada anteriormente, se indica que es una petición `GET` y que el tipo de contenido es `json`. Si el código de respuesta es diferente de `200 (OK)` se termina esta solicitud, de lo contrario se lee completamente la respuesta, la cual es una cadena `json` y se convierte a un objeto de tipo `iTunesClass`.

```

99 private void findiTunes(String filter) {
100     try {
101
102         URL url = new URL(REST_ITUNES_SERVICE_URL + filter);
103         HttpURLConnection conn = (HttpURLConnection) url.openConnection();
104         conn.setRequestMethod("GET");
105         conn.setRequestProperty("Accept", "application/json");
106         if (conn.getResponseCode() != 200) {
107             throw new RuntimeException("Failed : HTTP Error code : "
108                                     + conn.getResponseCode());
109         }
110         InputStreamReader in = new InputStreamReader(conn.getInputStream());
111         BufferedReader br = new BufferedReader(in);
112
113         StringBuilder output = new StringBuilder();
114         String line;
115
116         while ((line = br.readLine()) != null) {
117             output.append(line);
118         }
119
120         System.out.println(output.toString());
121
122         if (output.toString().isEmpty() == false)
123         {
124             Gson gson = new Gson();
125
126             iTunesClass results = gson.fromJson(output.toString(), new TypeToken<iTunesClass>() {}.getType());
127
128             if(results != null){
129                 for(resultsClass each: results.getResults()){
130                     //result += each.getResults().getTrackName() + "<br/>";
131
132                     list.add(new itemClass()
133                             .withType(each.getKind())
134                             .withName(each.getTrackName())
135                             .withUrl(each.getPreviewUrl())
136                             .withSummary(each.getLongDescription())
137                             );
138                 }
139             }
140         }
141         conn.disconnect();
142
143     } catch (Exception e) {
144         System.out.println("Exception in NetClientGet:- " + e);
145         result = FAIL + " - " + e;
146     }
147 }
148

```

La clase `iTunesClass` contiene una lista de objetos de la clase `resultsClass`.

```

1 package apiTest;
2
3 import java.util.List;
4
5 public class iTunesClass {
6     private float resultCount;
7     private List<resultsClass> results;
8
9     // Getter Methods
10
11     public float getScore() {
12         return resultCount;
13     }
14
15     public List<resultsClass> getResults() {
16         return results;
17     }
18
19     // Setter Methods
20
21     public void setResultCount( float resultCount ) {
22         this.resultCount = resultCount;
23     }
24
25     public void setResults( List<resultsClass> results ) {
26         this.results = results;
27     }
28
29 }
30

```

La clase `resultsClass` contiene la información que posteriormente se enviara en la respuesta de las coincidencias que cumple el criterio de búsqueda.

```

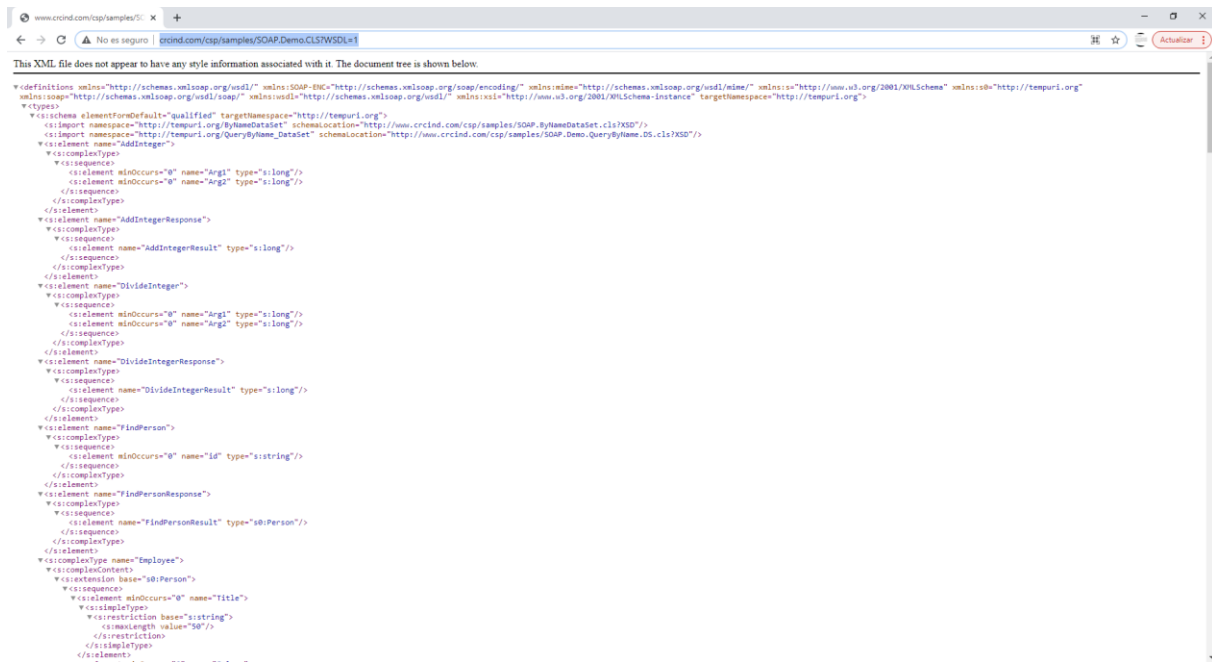
1 package apiTest;
2
3 public class resultsClass {
4     private float trackId;
5     private String previewUrl;
6     private String trackName;
7     private String kind;
8     private String longDescription;
9
10     public float getTrackId() {
11         return trackId;
12     }
13
14     public String getPreviewUrl() {
15         return previewUrl;
16     }
17
18     public String getTrackName() {
19         return trackName;
20     }
21
22     public String getKind() {
23         return kind;
24     }
25
26     public String getLongDescription() {
27         return longDescription;
28     }
29
30 }

```

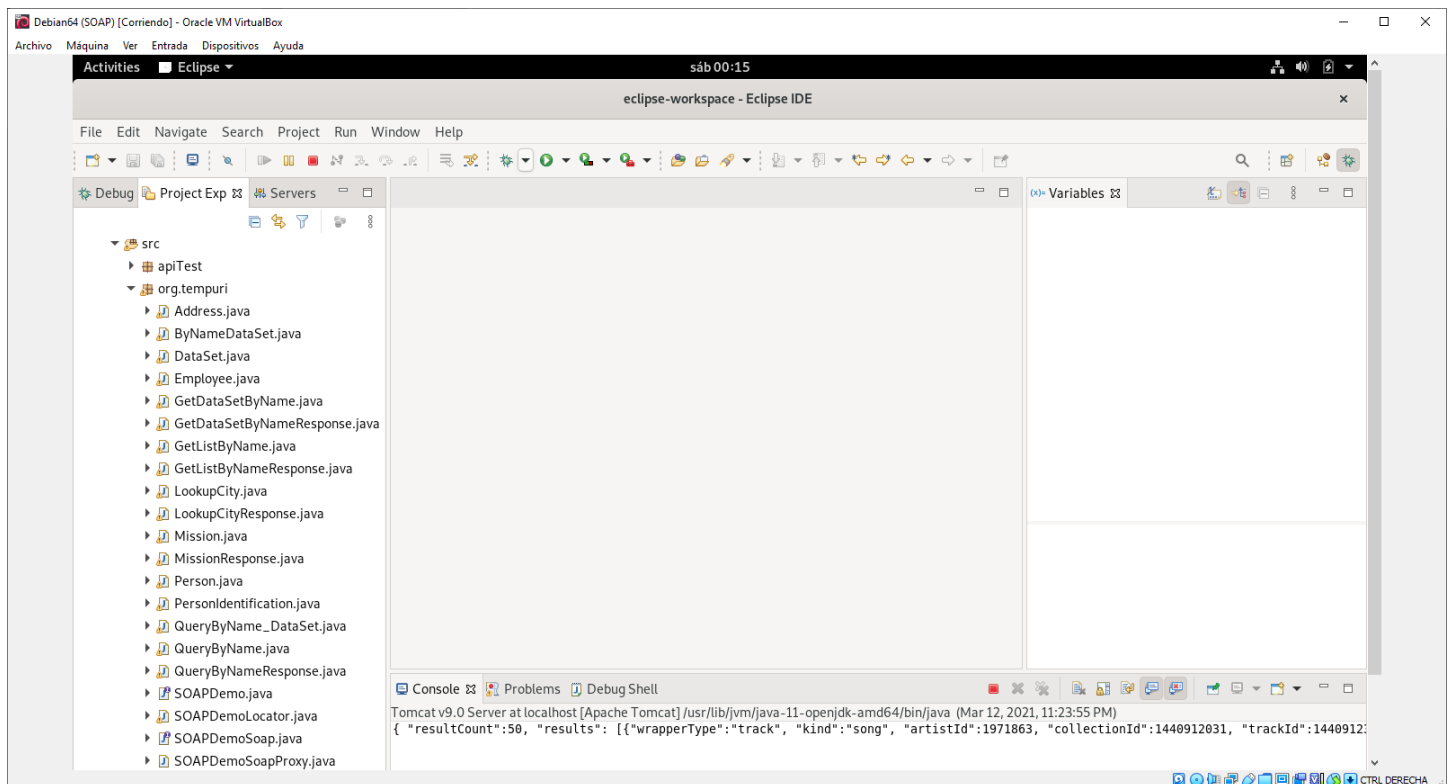
Para el caso del servicio para búsqueda de personas, se hizo la referencia al WSDL para que se crearan todas las clases que corresponden.

<http://www.crcind.com/csp/samples/SOAP.Demo.CLS?WSDL=1>





Se crearon todos los archivos java que utilizan los métodos y el servicio SOAPDemo.



Para realizar la búsqueda de personas se usa el metodo `findercind`. Se usa la Url mencionada anteriormente, se crea un nuevo cliente `SOAPDemoSoapProxy` y luego se obtiene una instancia `SOAPDemoSoap`. Finalmente se ejecuta el método `getListByName` enviando el parámetro de búsqueda. Si se obtiene respuesta y se tienen coincidencias, se agregan a la lista de resultados.

```

150 private void findcircind(String filter) {
151     SOAPDemoSoapProxy DemoSoapProxy = new SOAPDemoSoapProxy(REST_PERSON_SERVICE_URL);
152     SOAPDemoSoap DemoSoap = DemoSoapProxy.getSOAPDemoSoap();
153     try {
154         PersonIdentification[] personList = DemoSoap.getListByName(filter);
155
156         if (personList != null)
157         {
158             for(PersonIdentification each: personList)
159             {
160                 //result += each.getName() + "<br/>";
161
162                 list.add(new itemClass()
163                     .withType("person")
164                     .withName(each.getName())
165                     .withUrl(REST_PERSON_SERVICE_URL)
166                     .withSummary("SSN: " + each.getSSN() + " DOB: " + each.getDOB())
167                 );
168             }
169         }
170
171     } catch (Exception e) {
172         System.out.println(e);
173         result = FAIL + " - " + e;
174     }
175 }

```

Para devolver una única lista de resultados se creó la clase `responseClass` que contiene una cadena de respuesta y una lista de coincidencias que corresponden al parámetro de búsqueda.

```

1 package apiTest;
2
3 import java.util.List;
4
5 public class responseClass {
6     private String result;
7     private List<itemClass> list;
8
9     /**
10      * @return the result
11      */
12     public String getResult() {
13         return result;
14     }
15
16     /**
17      * @param result the result to set
18      */
19     public void setResult(String result) {
20         this.result = result;
21     }
22
23     /**
24      * @return the list
25      */
26     public List<itemClass> getList() {
27         return list;
28     }
29
30     /**
31      * @param list the list to set
32      */
33     public void setList(List<itemClass> list) {
34         this.list = list;
35     }
36 }

```

La clase `resultClass` contiene una lista de objetos de la clase `itemClass` que es la información homologada de las 3 fuentes de datos. Los atributos de la clase `itemClass` son:

Type (show,song, tv-episode, person)

Name

Url (es un Url que devuelve la fuente de datos, para el caso de persona siempre se devuelve la Url de la fuente de datos)

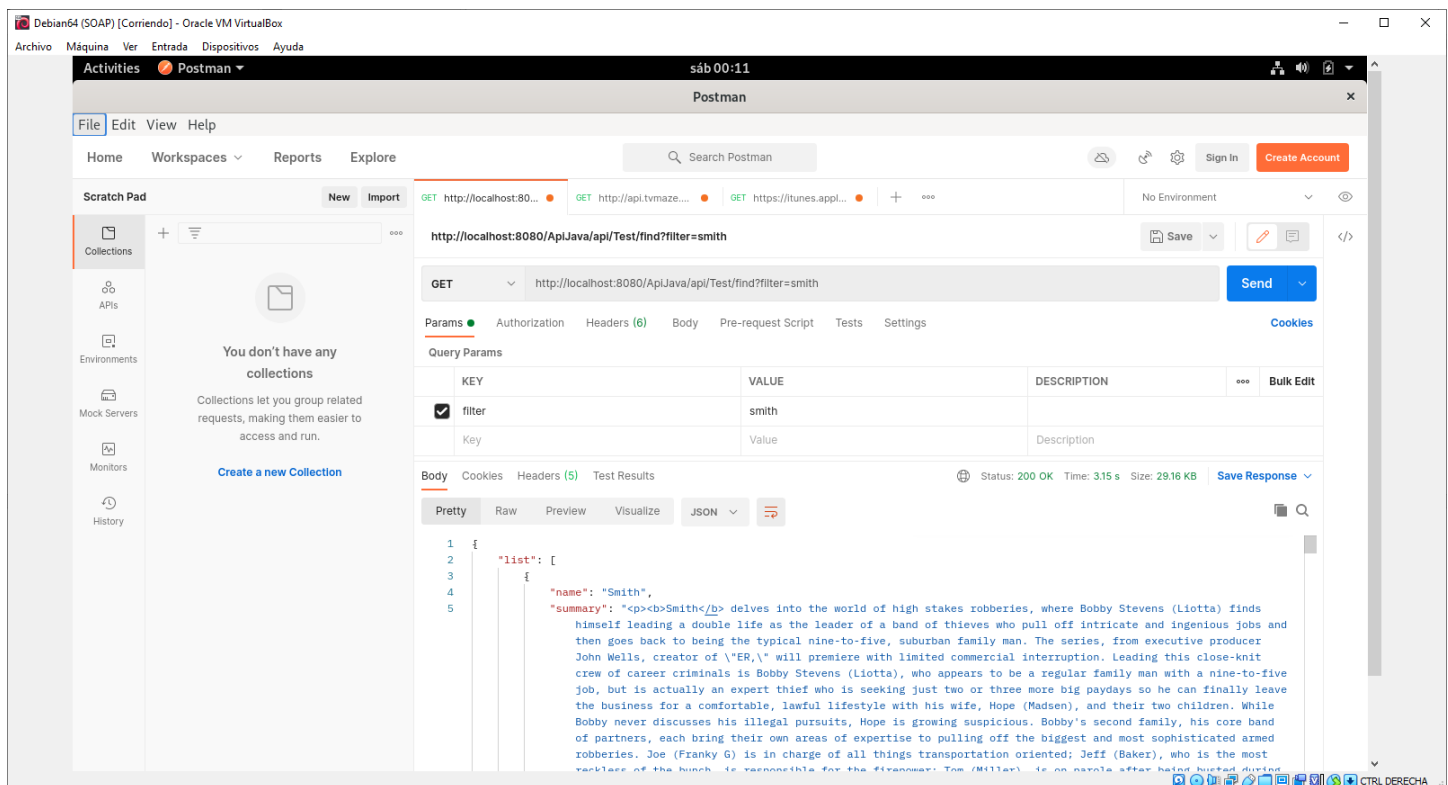
Summary (es una descripción del registro que contiene la fuente de datos)

```

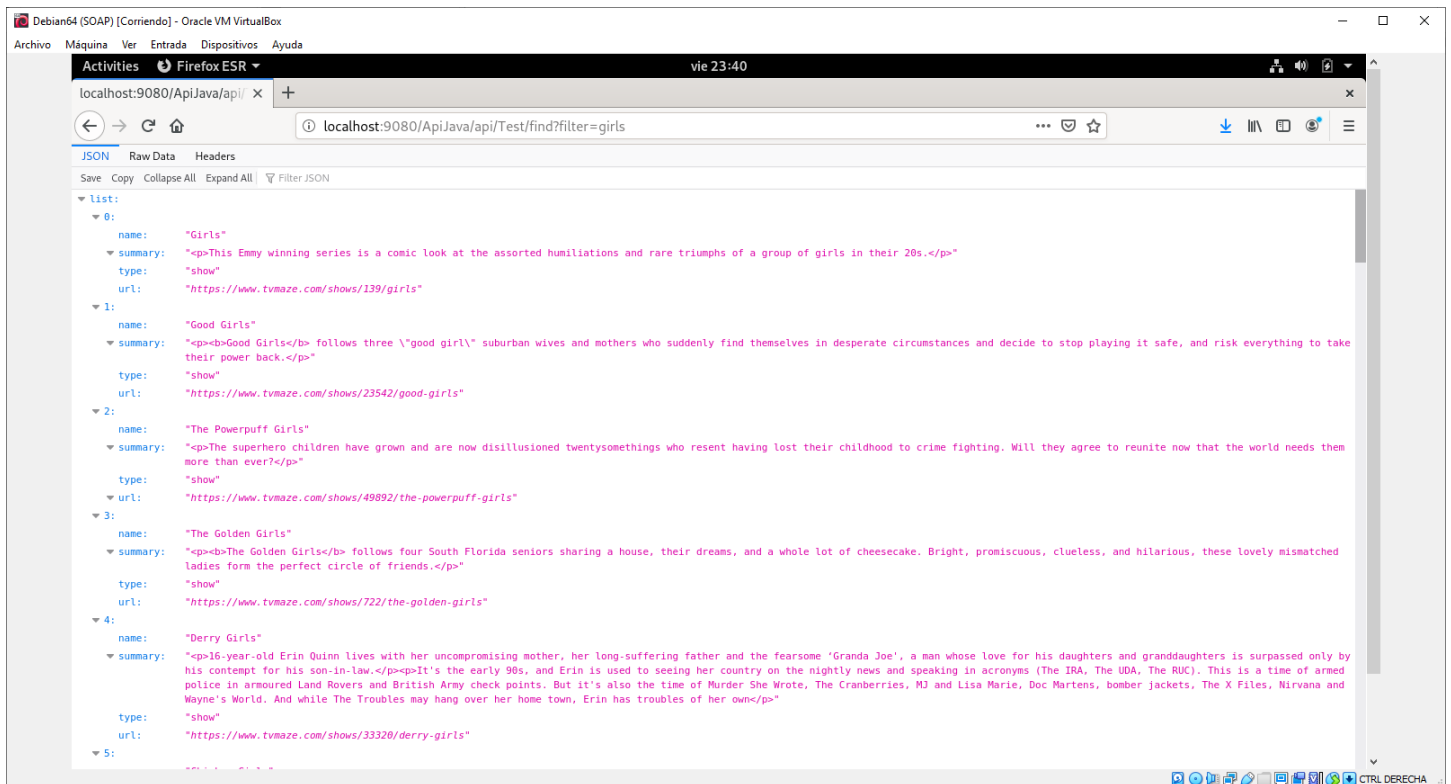
1 package apiTest;
2
3 public class itemClass {
4     private String type;
5     private String name;
6     private String url;
7     private String summary;
8
9     /**
10    *
11    */
12    public String getType() {
13        /**
14        */
15    }
16    public void setType(String type) {
17        /**
18        */
19    }
20    public String getName() {
21        /**
22        */
23    }
24    public void setName(String name) {
25        /**
26        */
27    }
28    public String getUrl() {
29        /**
30        */
31    }
32    public void setUrl(String url) {
33        /**
34        */
35    }
36    public String getSummary() {
37        /**
38        */
39    }
40    public void setSummary(String summary) {
41        /**
42        */
43    }
44    public itemClass withType(String type) {
45    }
46    public itemClass withName(String name) {
47    }
48    public itemClass withUrl(String url) {
49    }
50    public itemClass withSummary(String summary) {
51    }
52 }

```

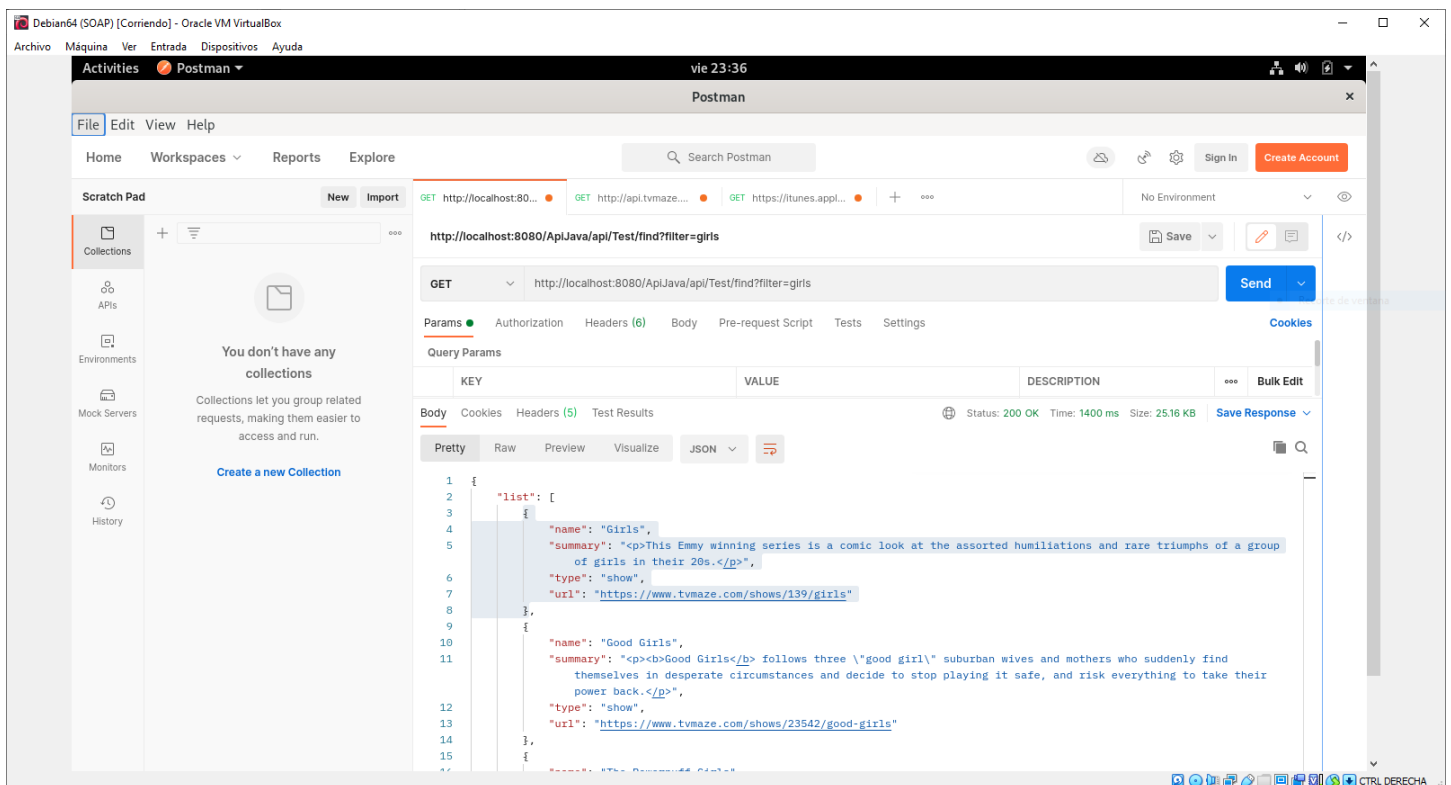
Finalmente se implementó en el servidor de aplicaciones en la dirección <http://localhost:8080/ApiJava/Test/find> y se debe de enviar el parámetro `filter` el criterio de búsqueda.



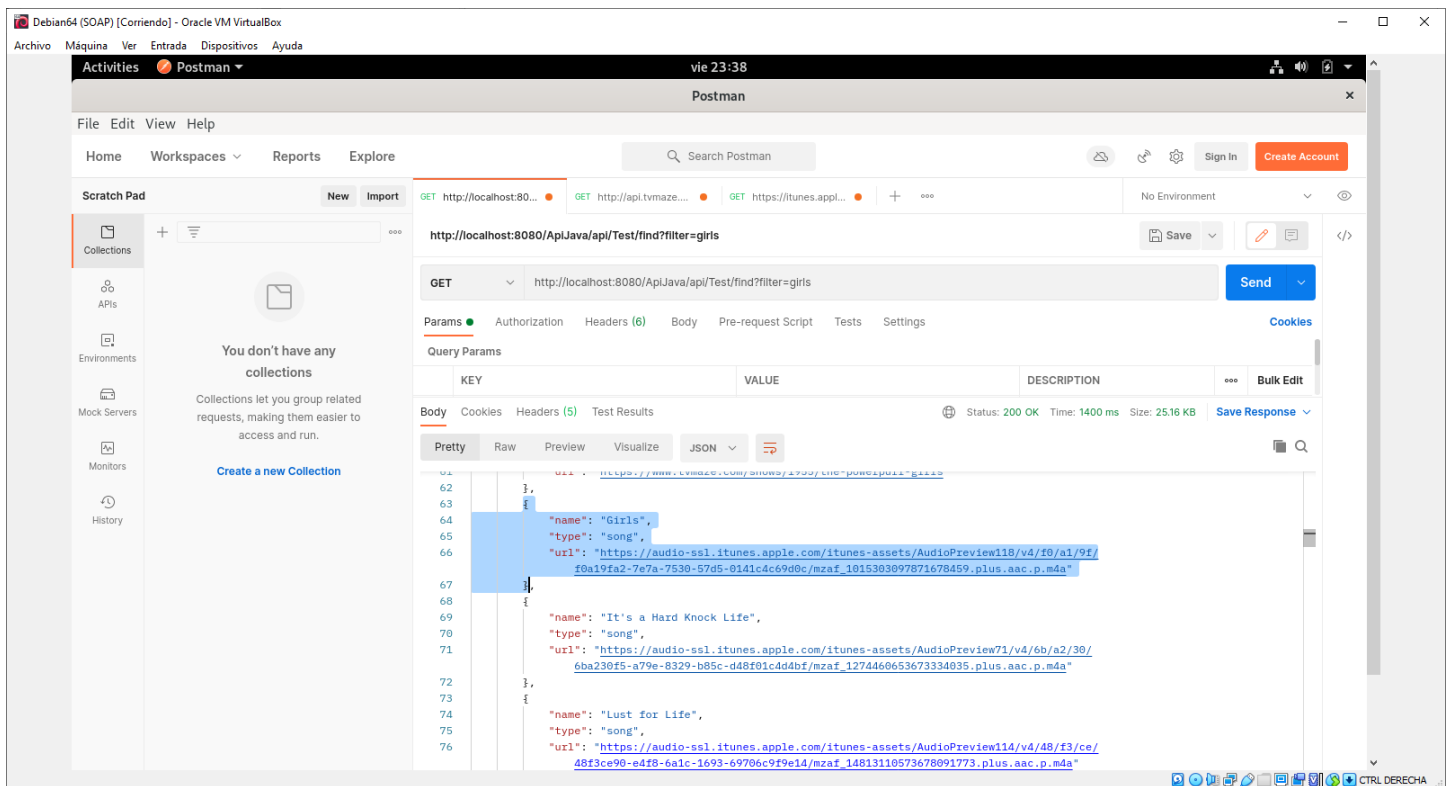
Ingresando desde un browser se muestra como la imagen a continuación.



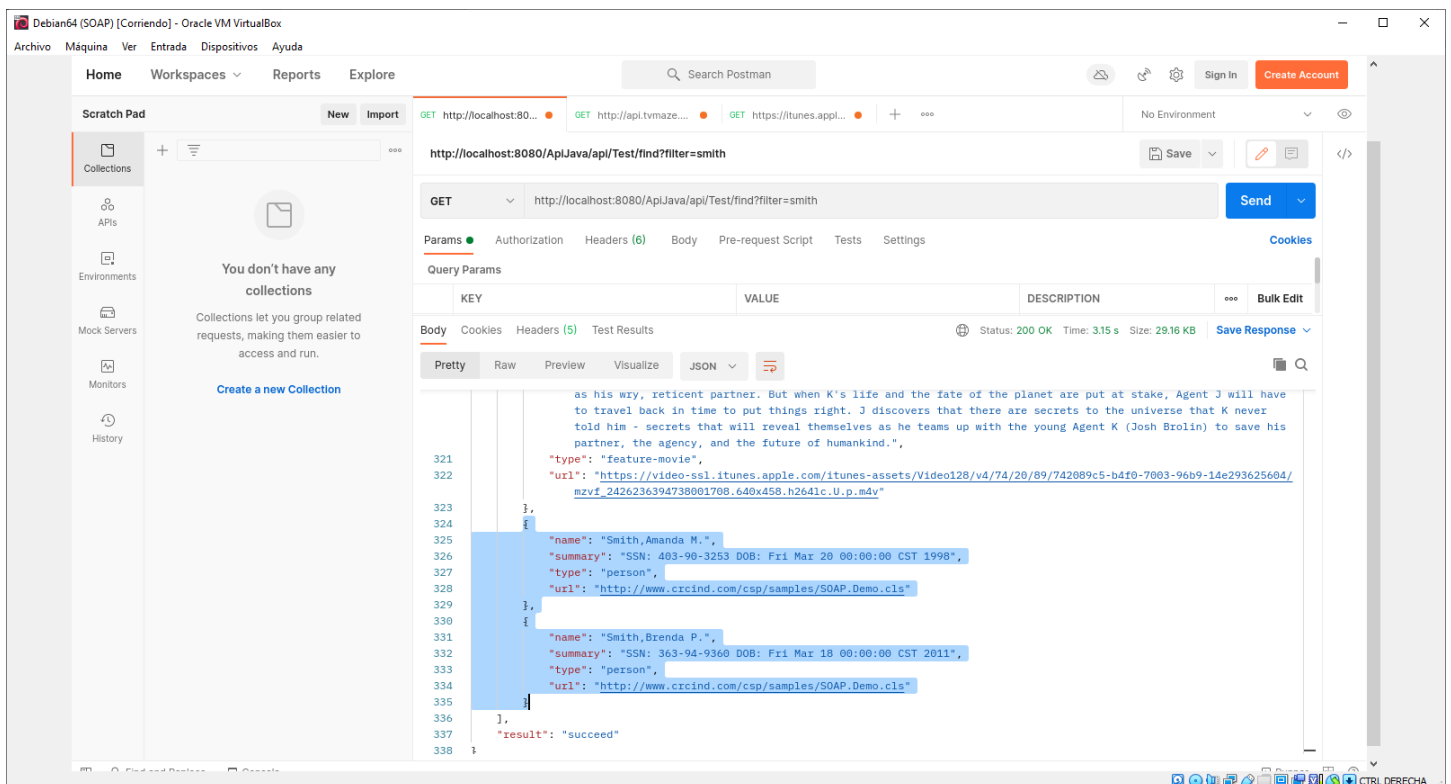
Así se devuelve los registros que corresponden a shows.



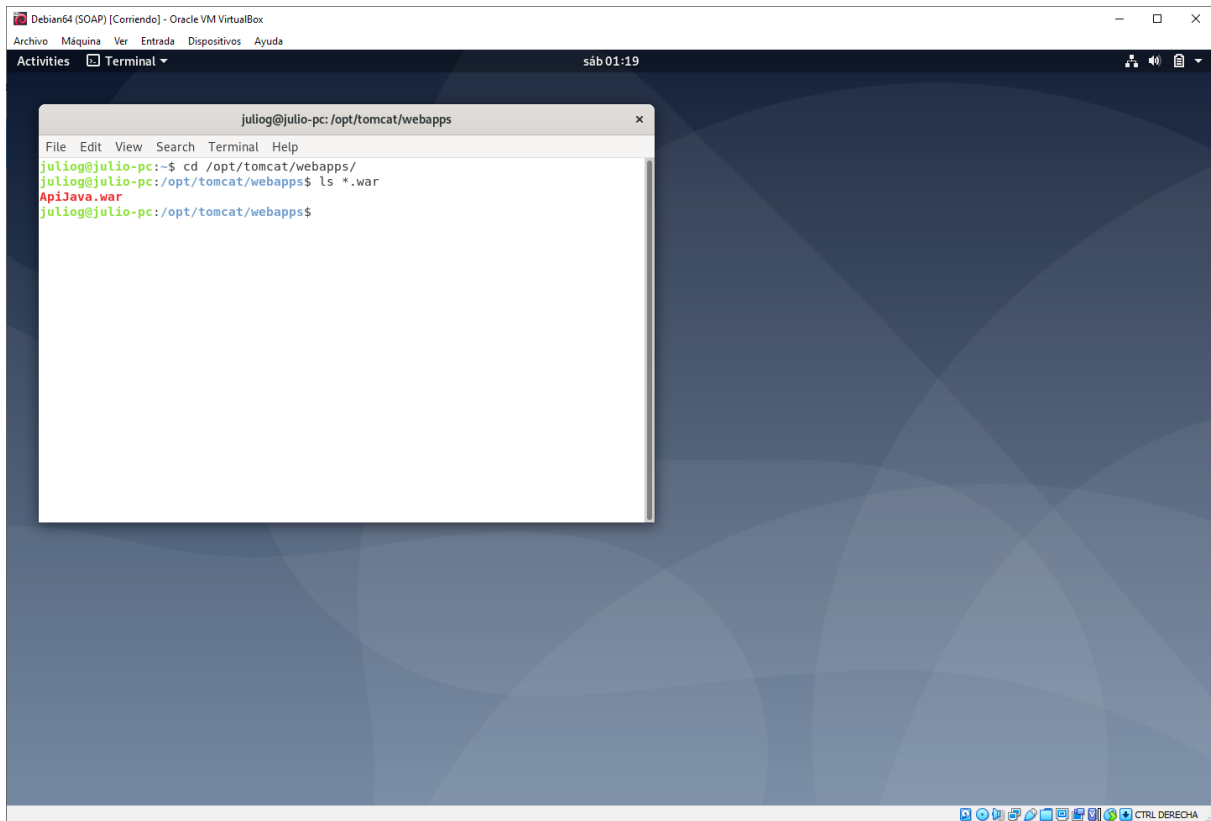
Así se devuelve los registros que corresponden a canciones y películas.



Así se devuelve los registros que corresponden a personas.



Para implementarlo únicamente es necesario copiar el archivo `ApiJava.war` en la carpeta `webapps` en el directorio donde se haya instalado Tomcat



The screenshot shows a terminal window titled "juliog@julio-pc: /opt/tomcat/webapps" within a Debian64 (SOAP) virtual machine. The terminal displays the following commands and output:

```
juliog@julio-pc:~$ cd /opt/tomcat/webapps/  
juliog@julio-pc:/opt/tomcat/webapps$ ls *.war  
ApiJava.war  
juliog@julio-pc:/opt/tomcat/webapps$
```