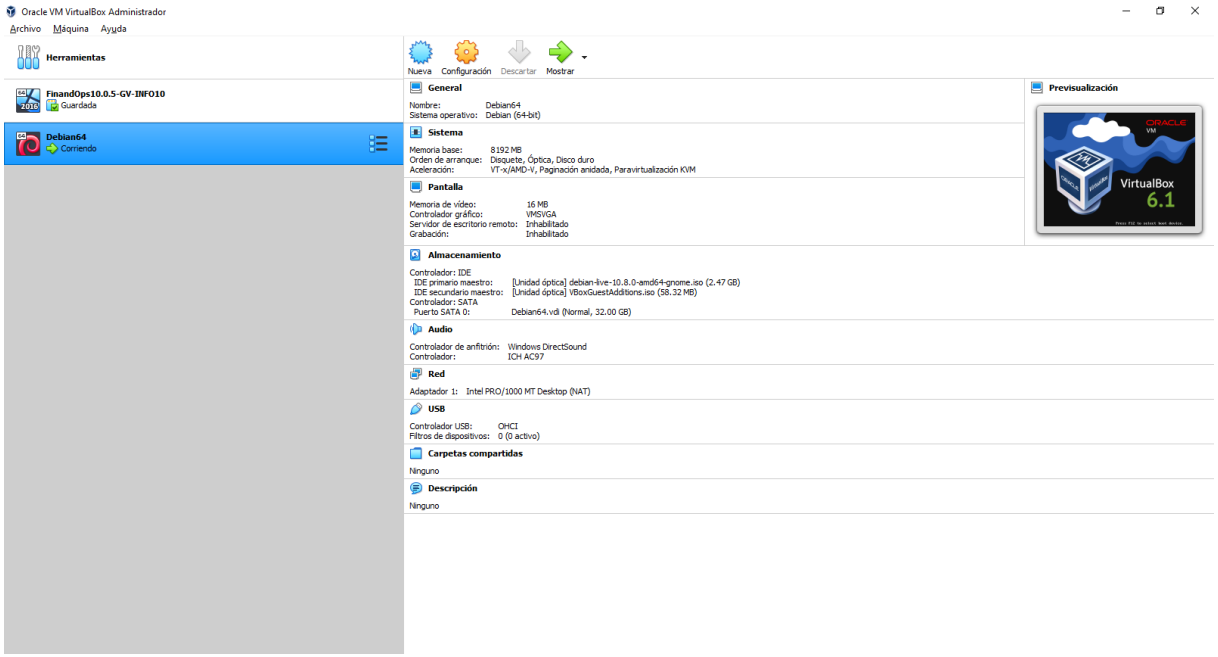
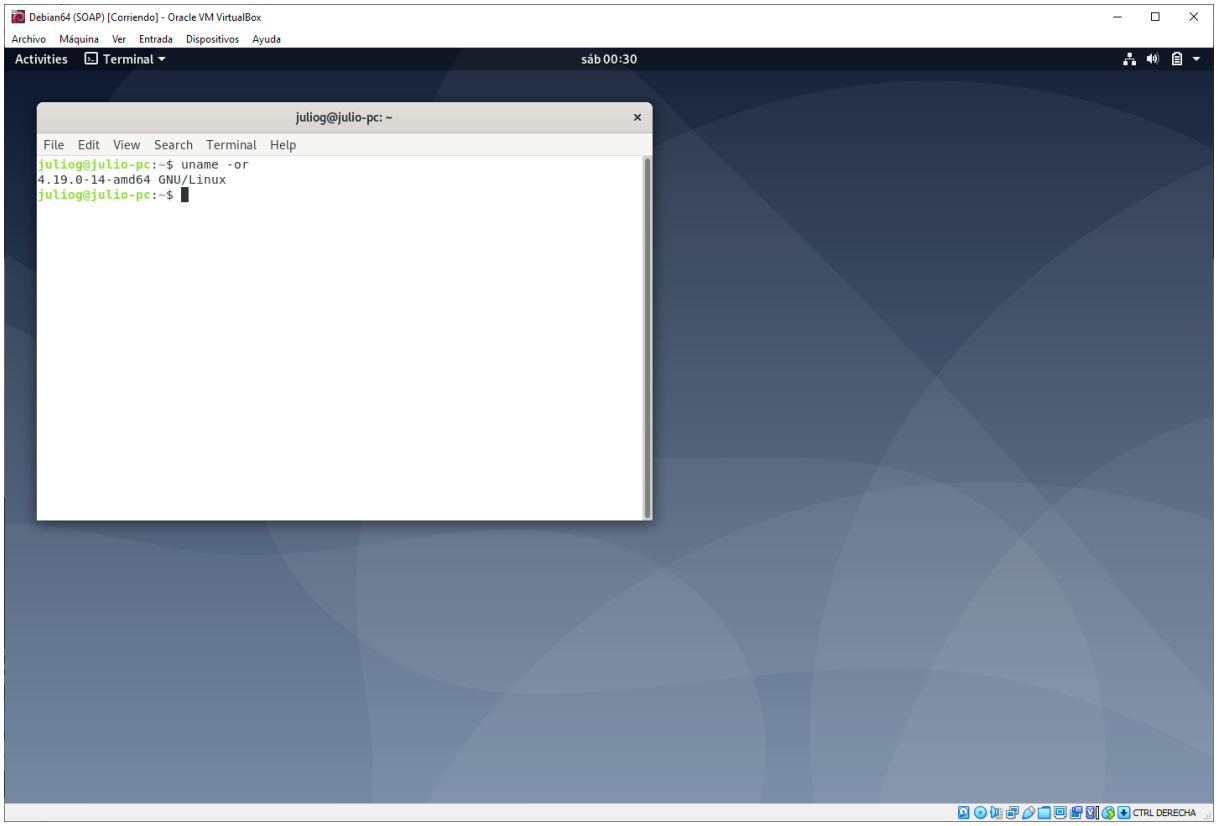
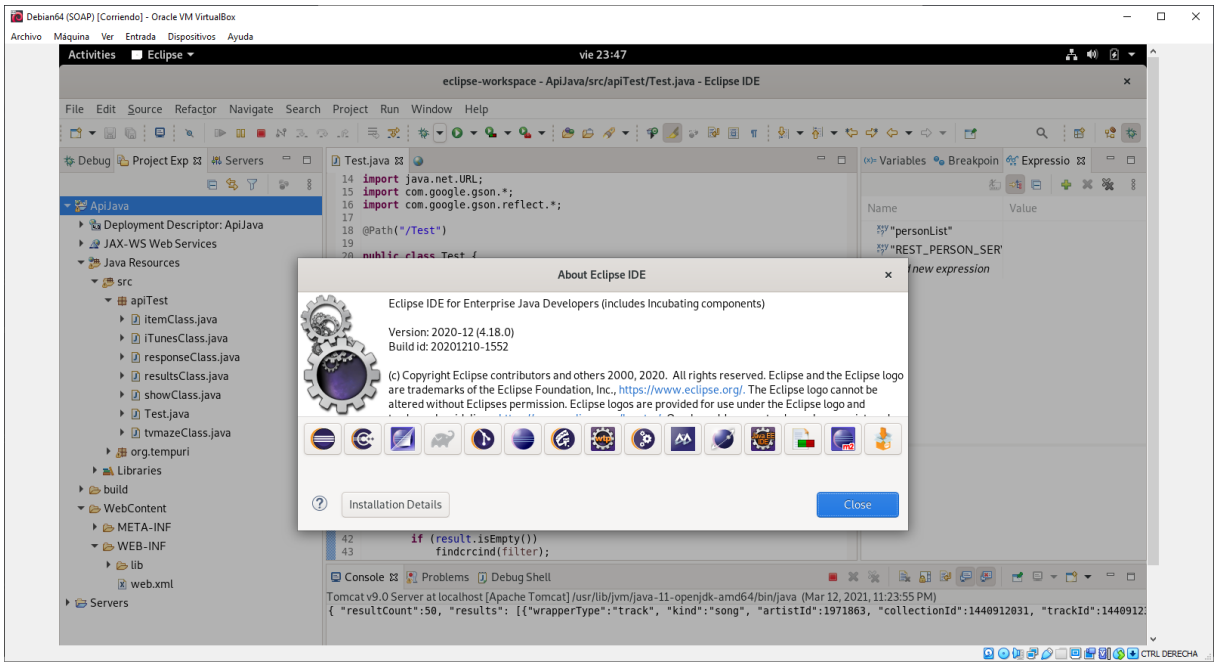
Se utilizó un Linux Debian 10 limpio en una máquina virtual.

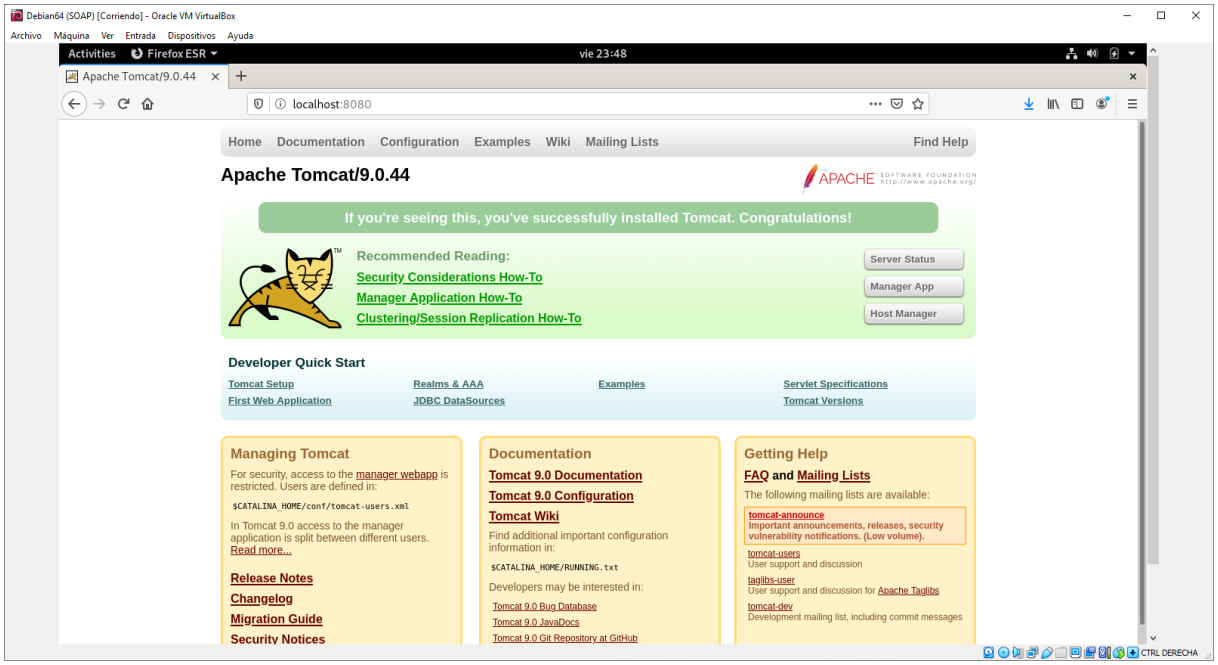




Como IDE se utilizó Eclipse

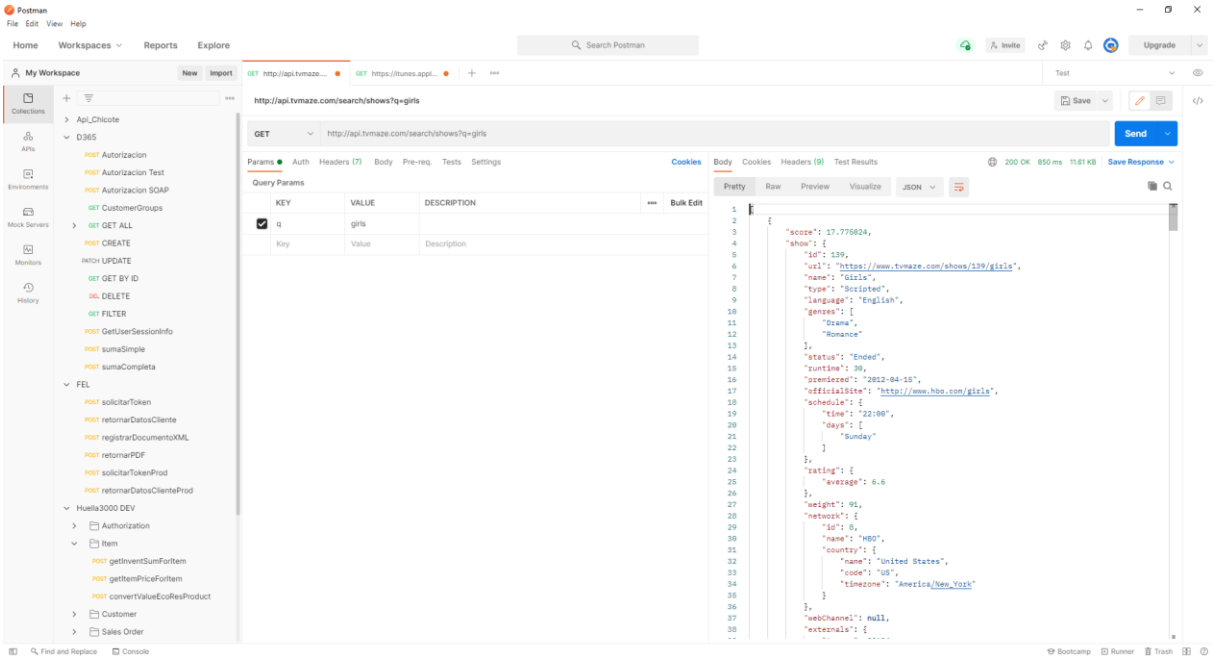


Como servidor de aplicaciones se utilizó Apache Tomcat v9.0.44

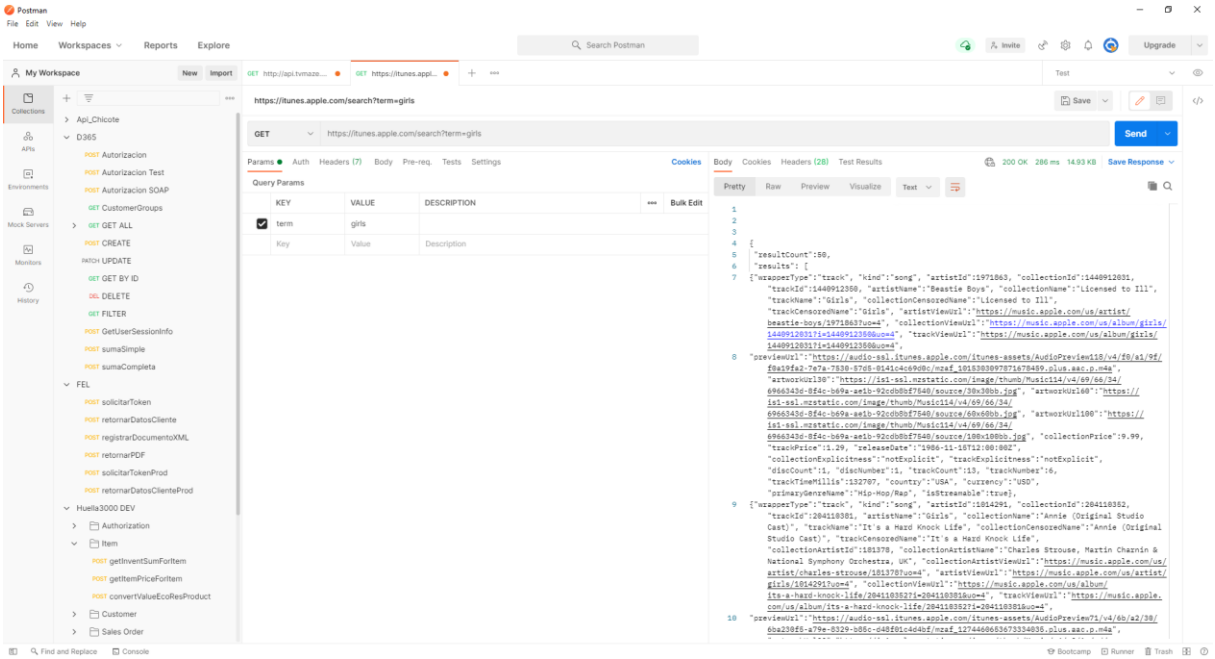


Se validó la respuesta de cada una de los 3 orígenes o fuentes de datos.

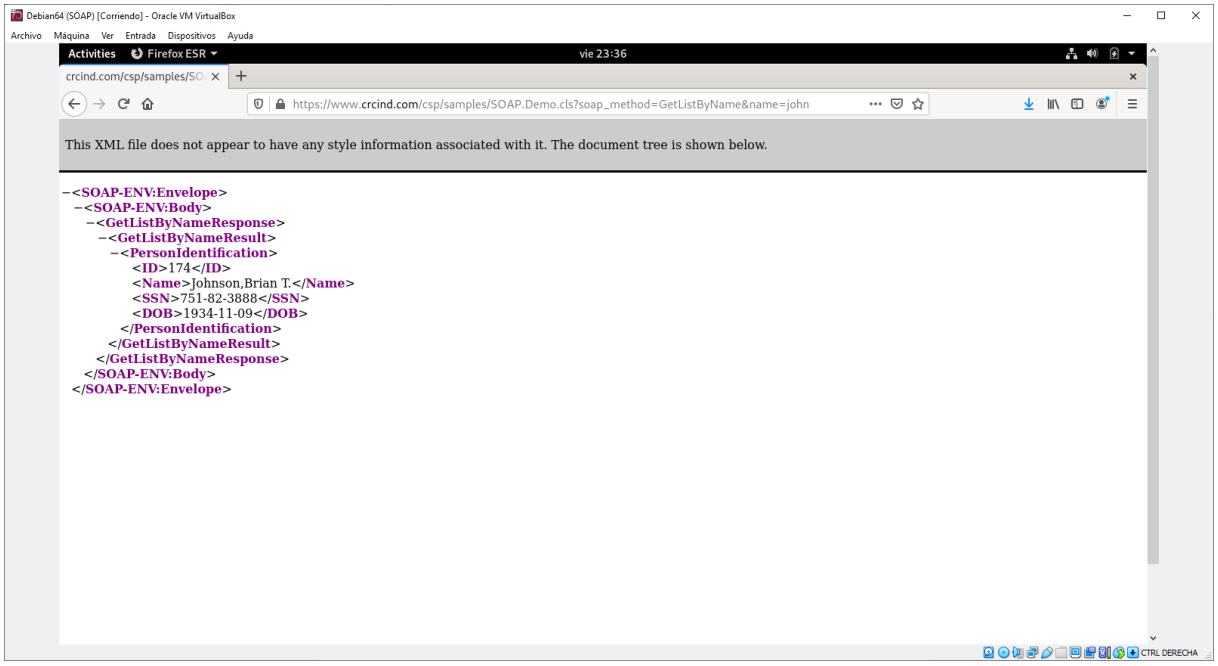
http://api.tvmaze.com/search/shows?q=girls **(REST)**



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



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
79                              //result += each.getShow().getName() + "<br/>";
80
81                              list.add(new itemClass()
82                                      .withType("show")
83                                      .withName(each.getShow().getName())
84                                      .withUrl(each.getShow().getUrl())
85                                      .withSummary(each.getShow().getSummary())
86                                      );
87                          }
88                      }
89                  }
90              }
91              conn.disconnect();
92
93          } catch (Exception e) {
94              System.out.println("Exception in NetClientGet:- " + e);
95              result = FAIL + " - " + e;
96          }
97      }
```

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.

```java
package apiTest;

public class showClass {
    private float id;
    private String url;
    private String name;
    private String type;
    private String summary;

    public float getId() {
        return id;
    }

    public String getUrl() {
        return url;
    }

    public String getName() {
        return name;
    }

    public String getType() {
        return type;
    }

    public String getSummary() {
        return summary;
    }
```

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
131                         //result += each.getResults().getTrackName() + "<br/>";
132
133                         list.add(new itemClass()
134                                 .withType(each.getKind())
135                                 .withName(each.getTrackName())
136                                 .withUrl(each.getPreviewUrl())
137                                 .withSummary(each.getLongDescription())
138                                 );
139                     }
140                 }
141             }
142             conn.disconnect();
143
144         } catch (Exception e) {
145             System.out.println("Exception in NetClientGet:- " + e);
146             result = FAIL + " - " + e;
147         }
148     }
```

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

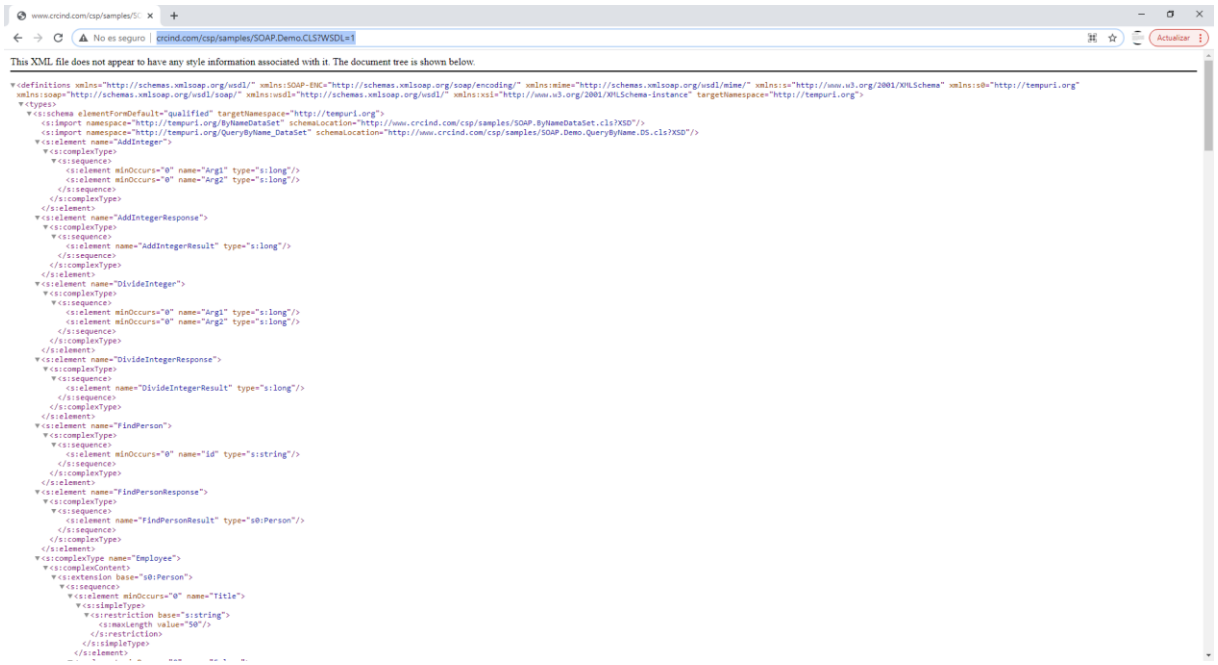
```java
package apiTest;

import java.util.List;

public class iTunesClass {
    private float resultCount;
    private List<resultsClass> results;

    // Getter Methods

    public float getScore() {
        return resultCount;
    }

    public List<resultsClass> getResults() {
        return results;
    }

    // Setter Methods

    public void setResultCount( float resultCount ) {
        this.resultCount = resultCount;
    }

    public void setResults( List<resultsClass> results ) {
        this.results = results;
    }

}
```

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

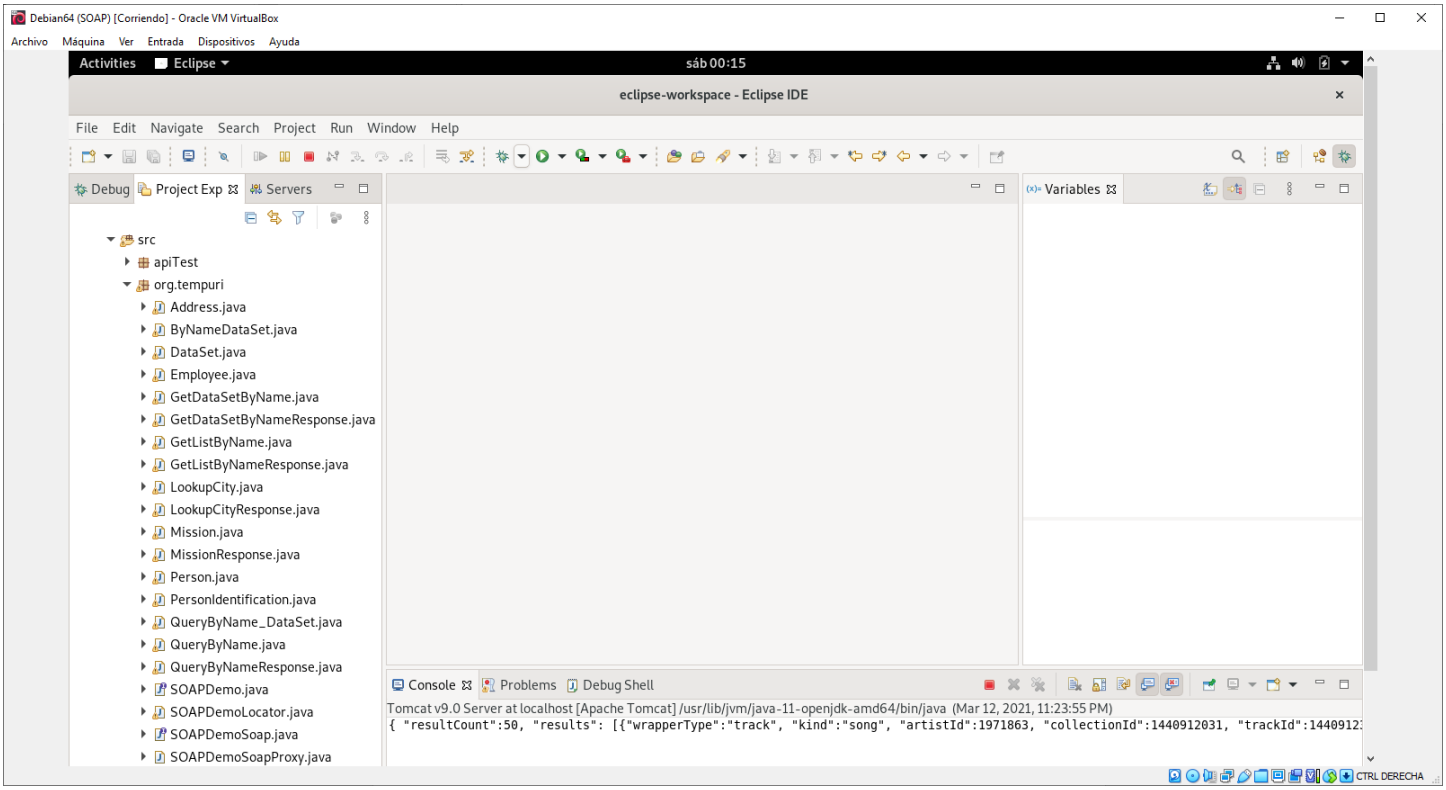
```java
package apiTest;

public class resultsClass {
    private float trackId;
    private String previewUrl;
    private String trackName;
    private String kind;
    private String longDescription;

    public float getTrackId() {
        return trackId;
    }

    public String getPreviewUrl() {
        return previewUrl;
    }

    public String getTrackName() {
        return trackName;
    }

    public String getKind() {
        return kind;
    }

    public String getLongDescription() {
        return longDescription;
    }

}
```

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 `findcrcind`. 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 findcrcind(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       * @param result the result to set
17       */
18      public void setResult(String result) {
19          this.result = result;
20      }
21      /**
22       * @return the list
23       */
24      public List<itemClass> getList() {
25          return list;
26      }
27      /**
28       * @param list the list to set
29       */
30      public void setList(List<itemClass> list) {
31          this.list = list;
32      }
33  }
```

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)`

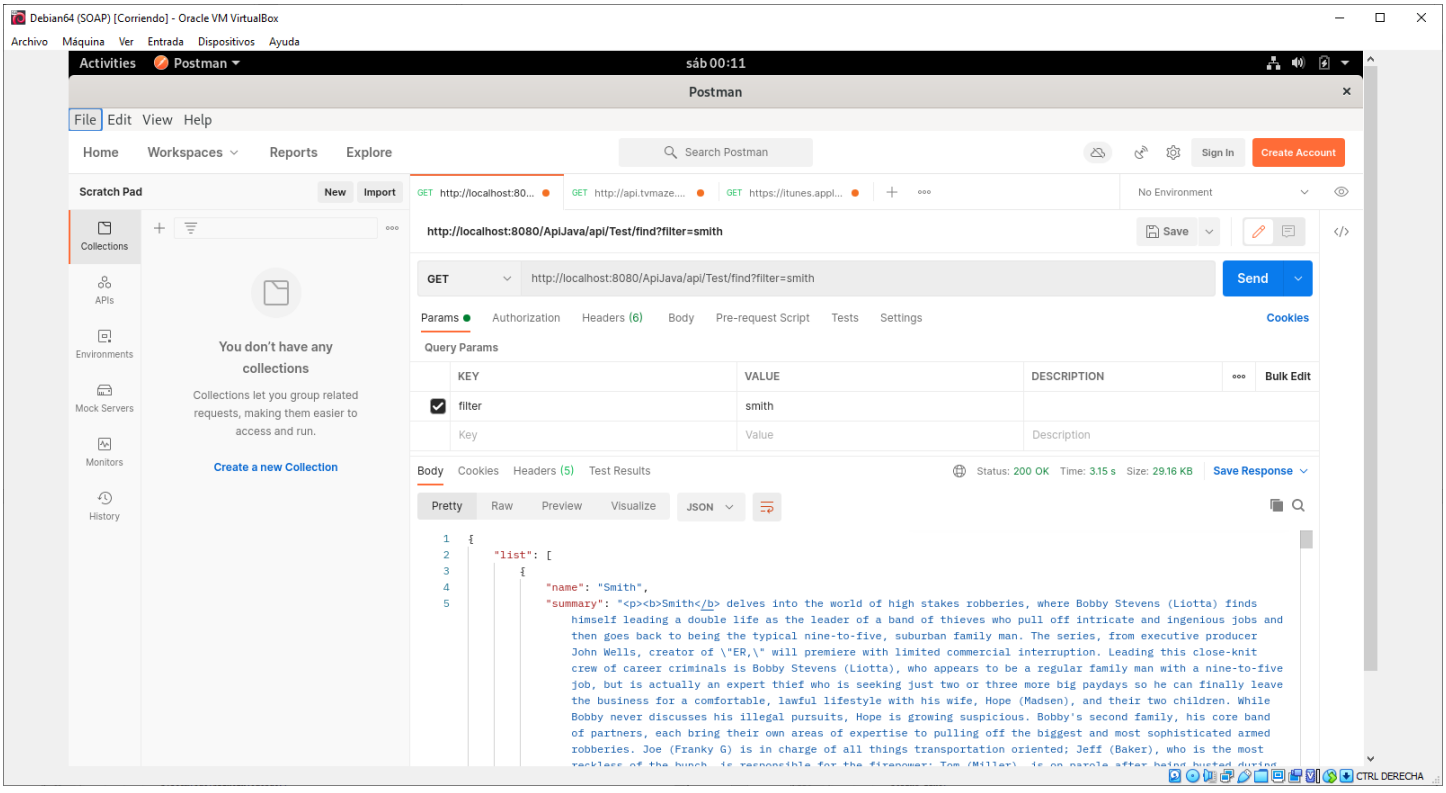
`Name`

`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)`

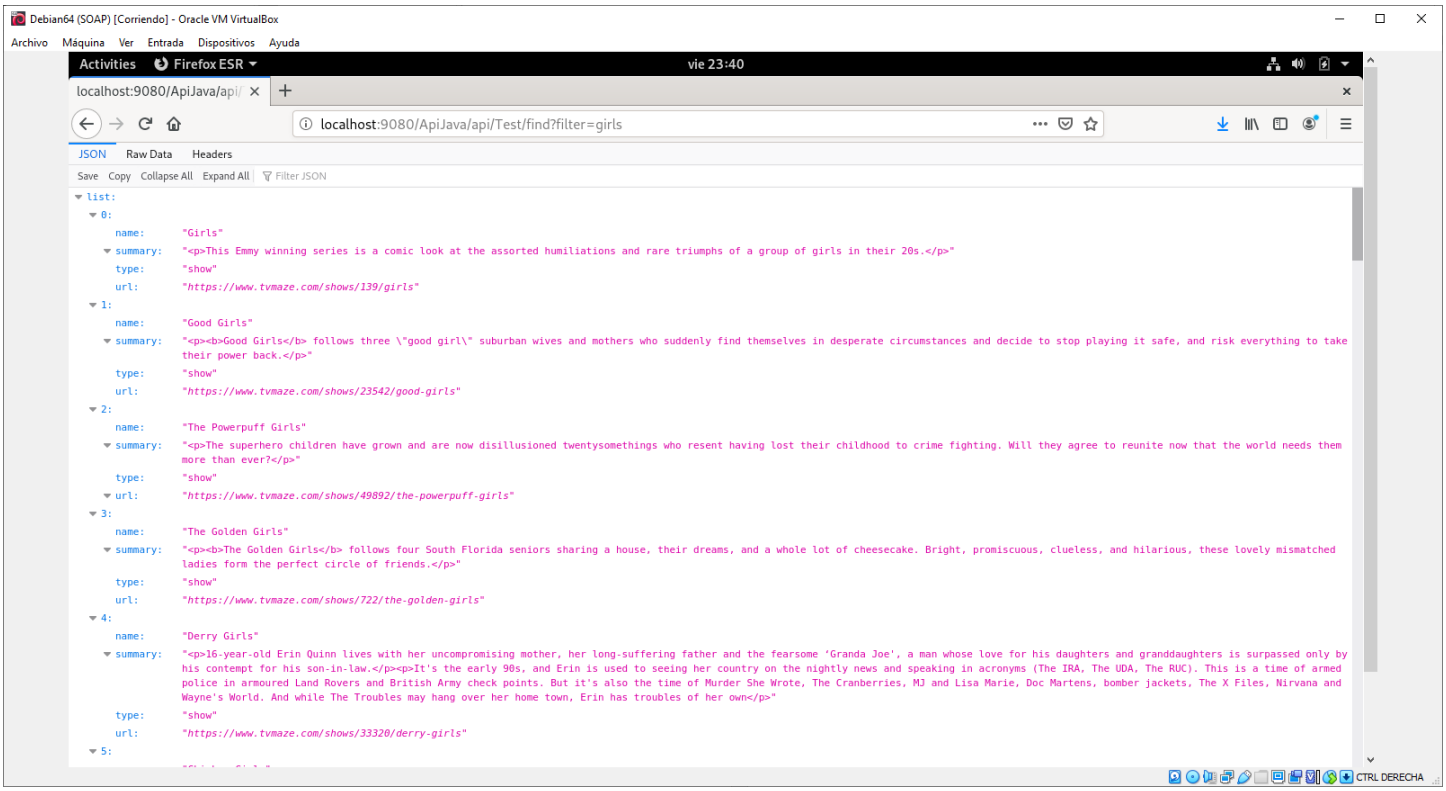
`Summary (es una descripción del registro que contiene la fuente de datos)`

```java
package apiTest;

public class itemClass {
    private String type;
    private String name;
    private String url;
    private String summary;

    /**
    public String getType() {
    /**
    public void setType(String type) {
    /**
    public String getName() {
    /**
    public void setName(String name) {
    /**
    public String getUrl() {
    /**
    public void setUrl(String url) {
    /**
    public String getSummary() {
    /**
    public void setSummary(String summary) {

    public itemClass withType(String type) {

    public itemClass withName(String name) {

    public itemClass withUrl(String url) {

    public itemClass withSummary(String summary) {
}
```

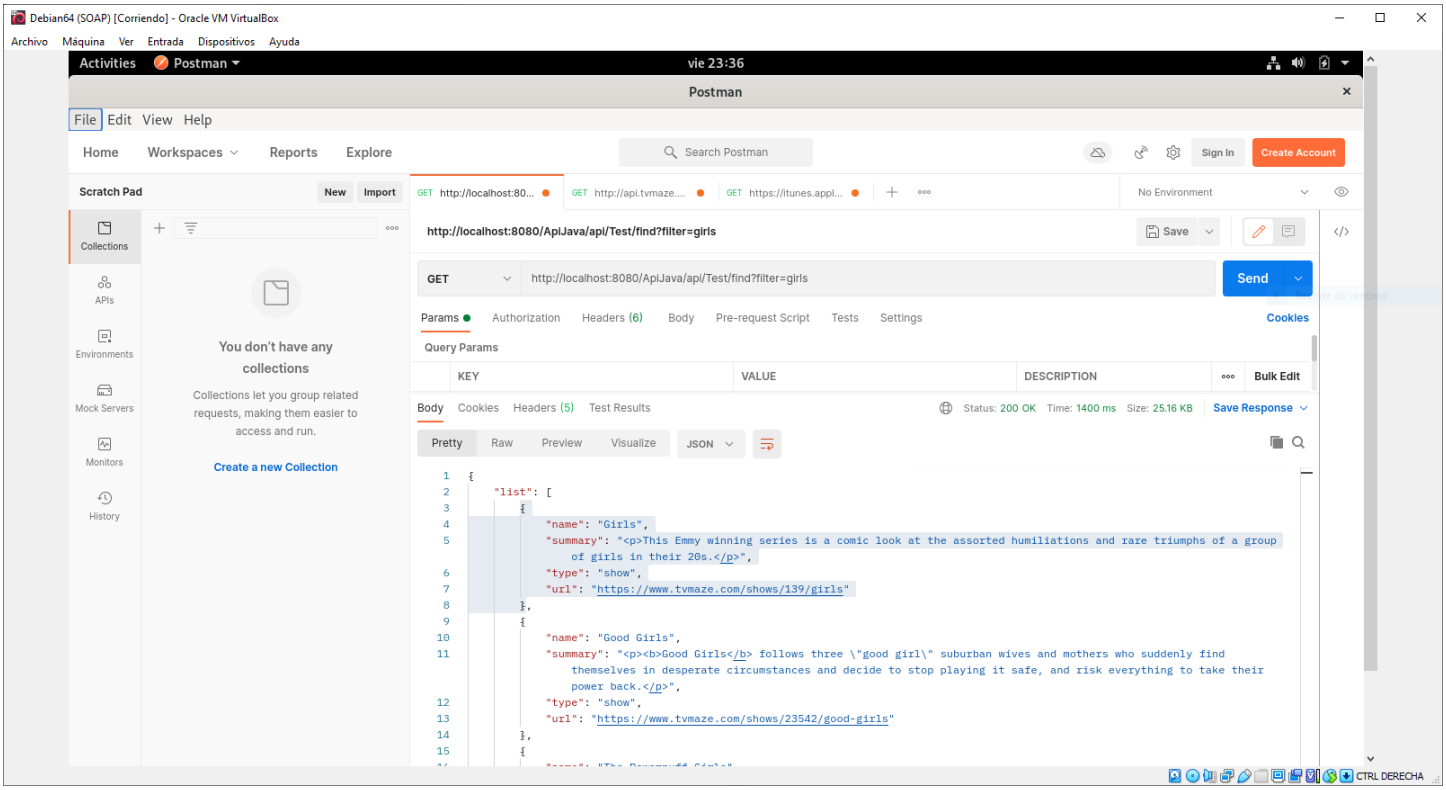
Finalmente se implementó en el servidor de aplicaciones en la dirección `http://localhost:8080/ApiJava/Test/find` y se debe de enviar el parámetro `filter` el criterio de búsqueda.



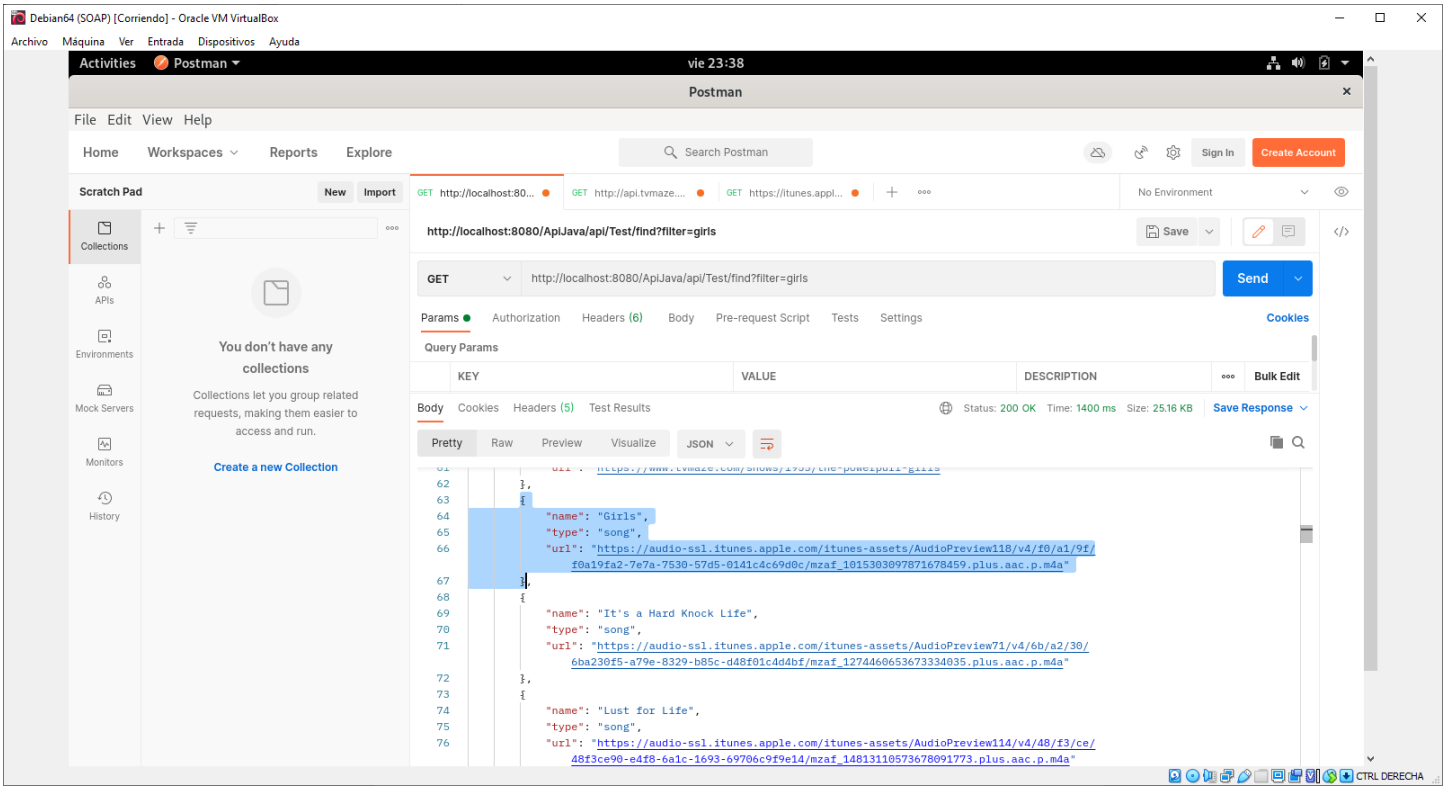
Ingresando desde un browser se muestra como la imagen a continuación.



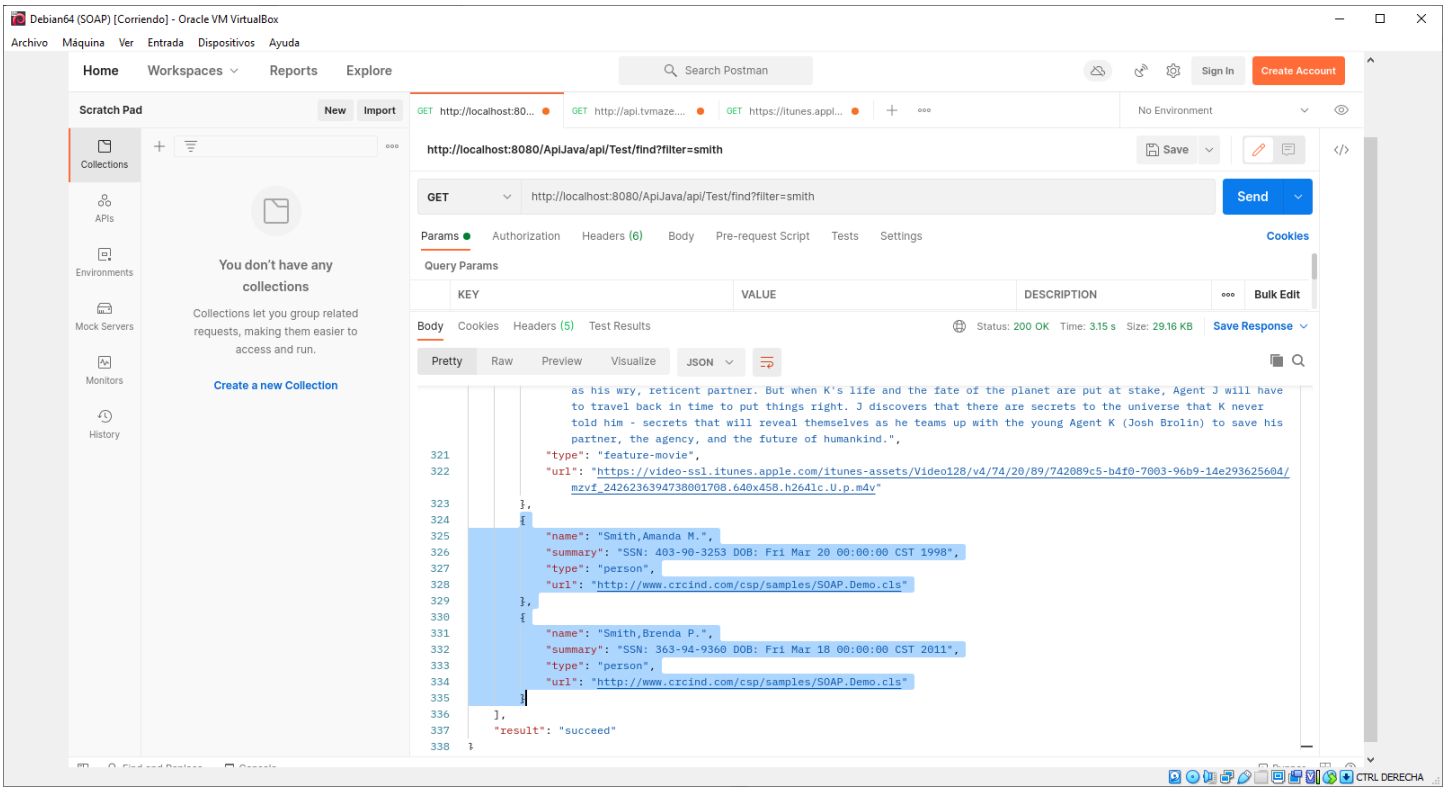
Así se devuelve los registros que corresponden a shows.



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`