Assignment-2

REST API for Actor/Actress

In this assignment, you are asked to develop a Flask-Restx data service that allows a client to read and store information about actors/actresses, and allows the consumers to access the data through a REST API.

The assignment is based on The TV Maze API, which provides a detailed list of TV shows and people. You can explore the TVmaze API using the following links

- The source URL: (http://api.worldbank.org/v2/) http://api.tvmaze.com/ (http://api.tvmaze.com/)
- (http://api.worldbank.org/v2/) Documentations on API Call Structure: https://www.tvmaze.com/api (https://www.tvmaze.com/api)

***Disclaimer: We are using an extremal API provided by TV Maze (https://www.tvmaze.com/). We want the students to interact with real-life web services to add to the learning experience and hence we are not responsible nor liable for the wording or inclusion/exclusion of entities and descriptions within the API. This is a "building your REST API" exercise and should be treated as such.

In this assignment, you are going to use the information provided in this API and add a few functionalities as listed below:

Assignment Specification

Question-1: Add a new Actor (3 marks)

This operation can be considered as an on-demand 'import' operation to get the details of an actor to store them in your application. The service will download the JSON data for the given actor/actress (by its name) ; You must use **SQLite** for storing the data (the name of the database should be **YOURZID.db**) locally after importing the actor .

You can use the following to query the API: https://api.tvmaze.com/search/people?q=? (https://api.tvmaze.com/search/people?q=?)
For example, you can check the following query: https://api.tvmaze.com/search/people?q=Brad%20Pitt (https://api.tvmaze.com/search/people?q=Brad%20Pitt)

To Add an actor, your API accepts a query parameter called "name".:

- **name** : name of an actor/actress

After importing the collection, the service should return a response containing at least the following information:

- **id** : a unique integer identifier automatically generated (this might be different than actor ID in tvmaze API)
- **last-update** : the time the collection is stored in the database
- **_links** : the URL with which the imported collection can be retrieved (as shown in below example)

***Important** : For this assignment , you are asked to access the given Web content programmatically. Some Web hosts do not allow their web pages to be accessed programmatically, some hosts may block your IP if you access their pages too many times. During the implementation, download a few test cases and access the content locally - try not to perform too many HTTP requests programmatically. Check the documentation of the tvmaze API to get insights about their rate-limiting policy.*

Example:

```
POST /actors?name=brad pitt
```

An example response [This is not what you store in DB, it is the call's response]

: 201 Created

```
{
    "id" : 123,
    "last-update": "2021-04-08-12:34:40",
    "_links": {
        "self": {
          "href": "http://[HOST_NAME]:[PORT]/actors/123"
        }
    }
}
```

- You must import an actor (e.g, Brad Pitt) if only the given name fully matches ("brad pitt", "brad-pitt", "Brad pitt" all match each other but they should not match a name like "bard pit"). Be noted that the TVMaze API provides a fuzzy search and hence tolerates typos, capital/small, dashes...etc. and at the same time provides more results than the exact name. You should only import the matching one (ignoring cases, and any character except the English alphabet, **SPACE,** and numbers) . If there are a few actors with the same exact name, you need to only import one of them (any)
  Hint: before sending the name to tvmaze API, replace all non-English characters (e.g., , -. _, ?) with Space and then send it to the API; as such "brad pit-t" will be converted to "brad pit t" and this will not match "brad pitt". Dot not make this very complicated; we are not expecting you handle all exceptions, etc. Keep is as simple as mentioned here in the spec.
- What and how you store the data in DB is up to you, but take a look at the rest of the questions to know what attributes you need to keep for each Actor. At least you need to store: "name", "country", "birthday", "deathday", "gender", and a list of his/her "shows". When you are importing an actor, you should also store all his/her "shows". For this, you need to explore "TVmaze" API to see how you can automatically get such a list.
- You should never change an ID of a resource (do not update ids)
- You should replace [HOST_NAME]:[PORT] with correct values
- A brief description about _links here: https://developer.wordpress.org/rest-api/using-the-rest-api/linking-and-embedding (https://developer.wordpress.org/rest-api/using-the-rest-api/linking-and-embedding/) /

Question 2 - Retrieve an Actor (1.5 marks)

This operation retrieves an actor by their ID (the ID that is generated by your application) . The response of this operation will show an actor's details. Please see the provided response example below to see what attributes should be included in the response. "_links" gives the links for previous, next, and current resources if they

exist . The next and previous resources are based on the sequential ID generated by your application.

The interface should look as like below:

```
GET /actors/{id}
```

Example Response returns: 200 OK

```
{
   "id": 124,
   "last-update": "2022-03-08-12:34:40",
   "name": "Some One",
   "country": "Australia",
   "birthday": "22-05-1987",
   "deathday": null,
   "shows": [
        "show 1",
        "show 2",
        "show 3"
    ],
  "_links": {
        "self": {
          "href": "http://[HOST_NAME]:[PORT]/actors/124"
        },
        "previous": {
          "href": "http://[HOST_NAME]:[PORT]/actors/123"
        },
        "next": {
          "href": "http://[HOST_NAME]:[PORT]/actors/125"
        }
      }
}
```

Question 3- Deleting an Actor (1.5 marks)

This operation deletes an existing actor from the database. The interface should look like as below:

```
DELETE /actors/{id}
```

Returns: 200 OK

```
{
    "message" :"The actor with id 134 was removed from the database!",
    "id": 134
}
```

Question 4 - Update an Actor (2 marks)

This operation partially updates the details of a given Actor.

The interface should look like the example below:

```
PATCH /actors/{id}
{
   "name": "Some One",
   "country": "Australia",
   "birthday": "22-05-1987",
   "deathday": null
}
```

The above payload is just an example; it can contain any of the actors' attributes. Take a look at the example response in Question 2 to know the existing attributes.

Returns: 200 OK

```
{
    "id" : 123,
    "last-update": "2021-04-08-12:34:50",
    "_links": {
        "self": {
          "href": "http://[HOST_NAME]:[PORT]/actors/123"
        }
    }
}
```

Question 5 - Retrieve the list of available Actors (4 marks)

This operation retrieves all available actors. The interface should be:

```
GET /actors?order=<CSV-FORMATED-VALUE> & page=1 & size=10 & filter=<CSV-FORMATED-VALUE>
```

All four parameters are optional with default values being "order=+id", "page=1", and "size =10", filter="id,name". "page" and "size" are used for pagination; "size" shows the number of actors per page. "order" is a comma-separated string value to sort the list based on the given criteria. The string consists of two parts: the first part is a special character '+' or '-' where '+' indicates ordering ascendingly, and '-' indicates ordering descendingly. The second part is an attribute name which is one of {id, name, country, birthday, deathday, last-update}. Here are some sample values of "order" :

+name,+id	order by "name ascending" and "id ascending" In this case, sorti ng by "name" has p riority over "id". This is similar to SQL order by clause : " name ASC, id ASC "
- country	order by "country descending"

"filter" is also another comma-separated value (id, name, country, birthday, deathday, last-update, shows), and shows what attribute should be shown for each actor accordingly. Take a look at the following example:

```
GET /actors?order=+id&page=1&size=10&filter=id,name
```

All four parameters are optional with default values being "order=+id", "page=1", and "size=10", "filter=id,name"

Returns: 200 OK

```
{
    "page": 1,
    "page-size": 10,
    "actors": [
           {
            "id" : 1,
            "name" : "Brad Pitt"
            },
           {
            "id" : 2,
            "name" : "Angelina Jolie"
           },
           ...
        ],
    "_links": {
        "self": {
          "href": "http://[HOST_NAME]:[PORT]/actors?order=+id&page=1&size=10&filter=id,name"
        },
        "next": {
          "href": "http://[HOST_NAME]:[PORT]/actors?order=+id&page=2&size=10&filter=id,name"
        }
      }
}
```

Question 6 - get the statistics of the existing Actors (3 marks)

This operation accepts a parameter called "format" which can be either "json" or "image". Depending on the format your operation should provide the following information: In case when the format is an image, your operation should return an image (can be in any image format) and the image illustrates the requested information in a visualization (apply all your knowledge when creating the visualization such as choosing appropriate visualization type and making sure that it is human-readable, clear, and informative).

- Actors break down by an attribute determined by the "by" parameter (a comma-separated value); this parameter can be any of the following Actors' attributes: "country" (showing the percentage of actors per country), "birthday", "gender", and "life_status" (represents what percentages of actors are alive). You may decide to use different types of charts based on the given attribute.
- Total Number of actors
- Total Number of actors updated in the last 24 hours
- UPDATE: Both parameters are required

The interface should look as below when the format is JSON:

```
GET /actors/statistics?format=json&by=country,gender
```

Returns: 200 OK

```
{
   "total": 1241,
   "total-updated": 24,
   "by-country" : { "Australia": 60.7, "USA": 19.2, ... },
   "by-gender" : { "Male": 48.6, "Femail": 51.4}
}
```

Notice:

- You can only use the libraries (latest versions) used in the Labs
- TVMAZE API is only used in Question 1 for importing a new actor. The rest of the operations rely on the existing actors in your local database
- There is no template code for this assignment, your submission should stick to the assignment specifications.
- Your submission will be marked manually.
- You should adhere to the best design guidelines for REST API mentioned in the lecture (e.g., appropriate responses based on JSON format with proper status codes, full API documentation: the generated Swagger should be fully self-explanatory with operation summary, parameter descriptions, default values).
- You should consider cases such as invalid titles or any invalid attempts to use the endpoint (e.g. If the input title doesn 't exist in the data source, return error 404)
- You should return appropriate responses (JSON) in case of already imported collections
- Your code must be implemented in **flask-restx** and automatically generate swagger doc for testing the endpoints.
- Your code must be executable in CSE machines
- Your code must not require installing any software (python libraries are fine)
- Your code must be written in Python 3.5 or newer versions.
- Your operations (API methods) must return appropriate responses in JSON format, and appropriate HTTP response code! e.g., **500 Internal Server Error** is an inappropriate response!
- Make sure you are using the right datatypes in the database and in your API methods (e.g., not string for years '2017')
- Some of the responses of some operations contain "_links", depending on the response this should include links to "self", "next", and "previous" resources.

FAQ:

- **What is the date-time format?**
  You can use any format (human-readable), but it should be the case for all date/time values, including birthday, last updated
- **How can I get an actor's SHOWs?**
  This is part of the assignment, and you should explore the API and find a way to link actors and shows
- **What libraries can I use?**
  You can use all libraries imported in any labs, plus SQLAlchemy
- **Can API users update ID, links, and Last-update time?**
  No, these attributes are not editable by the user. An ID should be never changed, links and last-update fields are not part of an actor's attribute.

Submission:

- The Deadline is **Monday the 4th of April 2022 20:00**
- One and only one Python script file named " YOUR_ZID .py" which contains your code
- **How I can submit my assignment?**
  Go to the assignment page click on the "Make Submission" tab; pick your files which must be named "YOUR_ZID.py". Make sure that the files are not empty, and submit the files together.
- **Can I submit my file after the deadline?**
  Yes, you can. But 5% of your assignment will be deducted as a late penalty per day. No late submission is accepted after 5 days.