# Assignment-2



# Data Service for TV Shows

In this assignment, you are asked to develop a Flask-Restx data service that allows a client to read and store some TV Shows, and allow 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. You can explore the TVmaze API using the following links*

- The source URL: (http://api.worldbank.org/v2/) http://api.tvmaze.com/shows (http://api.tvmaze.com/shows)
- (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 TV shows 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: import a TV Show (2 marks)

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

You can use the following to query the API: http://api.tvmaze.com/search/shows?q= (http://api.tvmaze.com/search/shows?q=TITLE) ?
For example, you can check the following query: http://api.tvmaze.com/search/shows?q=good%20girls (http://api.tvmaze.com/search/shows?q=good%20girls)

To import a tv show, your API accepts a query parameter called "name".:

- **name** : title for the tv show

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 tvmase_id)

- **tvmaze-id** : the id of the tv show in tvmaze API
- **last-update** : the time the collection 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 pages and access the content locally - try not to perform too many programmatically. Check the documentation of the tvmaze API to get insights about their rate limiting policy.*

*Example:*

```
POST /tv-shows/import?name=good girls
```

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",
    "tvmaze-id" : 23542,
    "_links": {
        "self": {
          "href": "http://[HOST_NAME]:[PORT]/tv-shows/123"
        }
    }
}
```

- You must import a TV show if only the given name matches a valid TV show (good girls, Good Girls, Good-Girls all match each other but they should not match a TV show like "Good Boys"). 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 TV Show. You should only import the matching one (ignoring cases, and any character except English alphabet and numbers ). In case there are more than one TV show with the same one, it is up to you how to deal with it (e.g., importing both, importing the latest one, etc)
- What and how you store the data in DB is up to you, but take a look at the rest of questions to know what attributes you need to keep for each TV show
- Do not get confused with having two identifiers ("id", and "tvmaze-id"); "id" is a unique identifier in your service and all of your operations relay on this id to work; "tvmaze-id" is just a reference to the original data.
- CLARIFICATION: You should never change an ID of a resource (do not update ids)
- CLARIFICATION: 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/) / <u></u>

---

## Question 2 - Retrieve a TV Show (2 marks)

This operation retrieves a collection by its ID (the ID that is generated by your application) . The response of this operation will show the details of TV show. 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 resource if they exist . The next and previous resources are based on the sequential ID generated by your application.

The interface should look like as like below:

```
GET /tv-shows/{id}
```

Example Response returns: 200 OK


```
{
   "tvmaze-id" :23542,
   "id": 124,
   "last-update": "2021-04-08-12:34:40",
   "name": "Good Girls",
   "type": "Scripted",
   "language": "English",
   "genres": [
        "Drama",
        "Comedy",
        "Crime"
    ],
    "status": "Running",
    "runtime": 60,
    "premiered": "2018-02-26",
    "officialSite": "https://www.nbc.com/good-girls",
    "schedule": {
        "time": "22:00",
        "days": [
          "Sunday"
        ]
     },
     "rating": {
        "average": 7.4
      },
      "weight": 100,
      "network": {
        "id": 1,
        "name": "NBC",
        "country": {
          "name": "United States",
          "code": "US",
          "timezone": "America/New_York"
        }
      },
      "summary": "<p><b>Good Girls</b> follows three \"good girl\" suburban wives and m
      "_links": {
        "self": {
          "href": "http://[HOST_NAME]:[PORT]/tv-shows/124"
        },
        "previous": {
          "href": "http://[HOST_NAME]:[PORT]/tv-shows/123"
        },
        "next": {
          "href": "http://[HOST_NAME]:[PORT]/tv-shows/125"
        }
      }
}
```

## Question 3- Deleting a TV show (2 marks)

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

```
DELETE /tv-shows/{id}
```

Returns: 200 OK

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

## Question 4 - Update a TV Show (2 marks)

This operation partially updates the details of a given TV Show.

The interface should look like the example below:

```
PATCH /tv-shows/{id}
{
   "name": "Good Girls",
   "language": "English",
   "genres": [
        "Drama",
        "Comedy",
        "Crime"
    ]
}
```

The above payload is just an example; it can contain any of the TV show 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]/tv-shows/123"
        }
    }
}
```

## Question 5 - Retrieve the list of available TV Shows 4 marks)

This operation retrieves all available TV shows. The interface should look like as like below:

```
GET /tv-shows?order_by=<CSV-FORMATED-VALUE> & page=1 & page_size=100 & filter=<CSV-FORM
```

All four parameters are optional with default values being "order_by=+id", "page=1", and "page_size =100", filter="id,name". "page" and "page_size" are used for pagination; "page_size" shows the number of TV shows per page. "order_by" 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,runtime,premiered,rating-average}. Here are some sample values of "order_by" :

| | |
|---|---|
| +rating-average,+id | order by "rating-average ascending" and then "id ascending"<br>In this case sorti ng by "rating-average" has p riority over "id". This is similar to SQL order by clause :<br>" rating-average ASC, id ASC " |
| -premiered | order by "premiered descending" |

"filter" is also another comma separated values (only consider= tvmaze_id ,id ,last-update ,name ,type ,language ,genres ,status ,runtime ,premiered ,officialSite ,schedule ,rating ,weight ,network ,summary), and show what attribute should be shown for each TV show accordingly. Take a look at the following example to know how response should be like:

```
GET /tv-shows?order_by=+id&page=1&page_size=100&filter=id,name
```

All four parameters are optional with default values being "order_by=+id", "page=1", and "page_size=100", "filer=id,name"

Returns: 200 OK

```
{
    "page": 1,
    "page-size": 100,
    "tv-shows": [
           {
            "id" : 1,
            "name" : "Good Girls"
            },
           {
            "id" : 2,
            "name" : "Brilliant Girls"
           },

           ...
        ],
    "_links": {
        "self": {
          "href": "http://[HOST_NAME]:[PORT]/tv-shows?order_by=+id&page=1&page_size=100
        },
        "next": {
          "href": "http://[HOST_NAME]:[PORT]/tv-shows?order_by=+id&page=2&page_size=100
        }
      }
}
```

## Question 6 - get the statistics of the existing TV Show (3 marks)

This operations 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 the format is 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).

- TV shows break down by an attribute determined by the "by" parameter; this parameter can be any of the following TV show attributes: "language" (showing the percentage of TV shows per Language), "genres", "status", and "type". In case of "genres", a TV show can have multiple values; you should come up with a solution to visualise it. For instance you can think of h *ow to visualise what percentage of movies belong to both "Comedy" and "Crime" genres, etc.*
- Total Number of TV shows
- Total Number of TV shows updated in the last 24 hours

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

```
GET /tv-shows/statistics?format=json&by=language
```

Returns: 200 OK

```
{
   "total": 1241,
   "total-updated": 24,
   "values" : { "English": 60.7, "French": 19.2, ... }
}
```

---

# Notice:

- TVMAZE API is only used in Question 1 for importing a new TV show. The rest of operations relay on the existing TV Shows in your local database
- There is not template code for this assignment, you 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 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 inappropriate response!
- Make sure you are using right datatypes in the database and in you API methods (e.g., not string for years '2017')
- Some of the response of some operations contain "_links", depending on the response this should include links to "self", "next", and "previous" resources.
-