Webservices API

API documentation for the webservice assignment

Movies : operations on movies

Movies

Request - GET

GET /movies/

> ***movieid**: integer*
> *The id of the movie.*

Response - GET

200

Returns information about the movie by its id.
The movie information is represented with following keys:

- ***id***: integer - The ID of the movie.
- ***title***: string - The original title of the movie.
- ***liked***: boolean - Whether the movie is liked by the user or not.

Example

```
>>> GET http://127.0.0.1:5000/movies/10597
```

```
{
    "id": 10597,
    "title": "Kevin & Perry Go Large",
    "liked": false
}
```

404

An error message for indicating there went something wrong.

Request - PUT

PUT /movies/

> ***movieid**: integer*
> *The id of the movie.*

Response - PUT

200

Likes or unlikes a movie based on the current like status of the movie.
Returns a little message when it succeeds.

Example

```
>>> PUT http://127.0.0.1:5000/movies/10597
```

```
"The movie with movie id 10597, is successfully liked/unliked."
```

404

An error message for indicating there went something wrong.

Request - DELETE

DELETE /movies/

> ***movieid**: integer*
> *The id of the movie.*

Response - DELETE

200

Deletes a mvoie by its id. Returns a little message when its succeed.

Example

```
>>> DELETE http://127.0.0.1:5000/movies/10597
```

```
"The movie with movie id 10597, is successfully deleted."
```

404

An error message for indicating there went something wrong.

Popular

Request

GET /movies/popular

> ***amount****: integer*
> *The number of popular movies to retrieve.*
> *Example value: 10*
> *Default value: 20*

Response

200

Returns a dictionary containing the popular movies.
Each movie's information is represented with following keys:

- ***id***: integer - The ID of the movie.
- ***title***: string - The original title of the movie.
- ***liked***: boolean - Whether the movie is liked by the user or not.

Example

```
>>> GET http://127.0.0.1:5000/movies/popular?amount=x
```

```
{
    "0":{
        "id": 10597,
        "title": "Kevin & Perry Go Large",
        "liked": false
    },

    "...": {},

    "x":{
        "id": 76600,
        "title": "Avatar: The Way of Water",
        "liked": false
    }
}
```


404

An error message for indicating there went something wrong.

Same genres

Request

GET /movies//same-genres

> ***movieid****: integer*
> *The id of the movie.*

Response

200

Returns a dictionary containing movies with the same genres as the provided movie.
Each movie's information is represented with following keys:

- ***id***: integer - The ID of the movie.
- ***title***: string - The original title of the movie.
- ***liked***: boolean - Whether the movie is liked by the user or not.

Example

```
>>> GET http://127.0.0.1:5000/movies/10597/same-genres
```

```
{
    "0":{
        "id": 76600,
        "title": "Avatar: The Way of Water",
        "liked": false
    },

    "...": {},

    "19":{
        "id": 12345,
        "title": "Avatar",
        "liked": false
    }
}
```

404

An error message for indicating there went something wrong.

Similar runtime

Request

GET /movies//similar-runtime

> ***movieid****: integer*
> *The id of the movie.*

Response

200

Returns a dictionary containing movies with the similar runtime as the provided movie.

Each movie's information is represented with following keys:

- ***id***: integer - The ID of the movie.
- ***title***: string - The original title of the movie.
- ***liked***: boolean - Whether the movie is liked by the user or not.

Example

```
>>> GET http://127.0.0.1:5000/movies/76600/similar-runtime
```

```
{
    "0":{
        "id": 76600,
        "title": "Avatar: The Way of Water",
        "liked": false
    },

    "...": {},

    "19":{
        "id": 67006,
        "title": "Star Wars: Episode IV: A New Hope,",
        "liked": false
    }
}
```

404

An error message for indicating there went something wrong.

Two overlapping actors

Request

GET /movies//overlapping-actors

> ***movieid****: integer*
> *The id of the movie.*

Response

200

Returns a dictionary containing movies with two overlapping actors as the provided movie.
Each movie's information is represented with following keys:

- ***id***: integer - The ID of the movie.
- ***title***: string - The original title of the movie.
- ***liked***: boolean - Whether the movie is liked by the user or not.

Example

```
>>> GET http://127.0.0.1:5000/movies/67006/overlapping-actors
```

```
{
    "0":{
        "id": 67006,
        "title": "Star Wars: Episode IV: A New Hope",
        "liked": false
    },

    "...": {},

    "19":{
        "id": 67005,
        "title": "Star Wars: Episode III: Revenge of the Sith",
        "liked": true
    }
}
```

404

An error message for indicating there went something wrong.

Comparing movies

GET /movies/compare

> ***movies**: comma-separated string contains ID as integers*
> *Is a comma-separated string containging the IDs of the movies to retrieve chart data for.*

Response

200

Returns a dictionary that contains data for plotting a chart for the requested movies. The dictionary contains the following keys:

- ***type***: string - The type of chart being returned. Currently 'bar'.
- ***data***: dictionary - Containing the data for the chart.
    - ***labels***: list - The movie names for the requested movies.
    - ***datasets***: list - Containing a single dictionary representing the dataset.
        - ***label***: string - The label for the dataset. Currently set to 'Voting average'.
        - ***data***: list - The average voting scores for the requested movies.

For plotting the chart in your browser you can paste the retrieved dictionary behind the argument 'c' as follows:

```
https://quickchart.io/chart?c=<dictionary>
```

Example

```
>>> GET http://127.0.0.1:5000/movies/compare?movies=67060,456123,74185
```

```
{
        "type": "bar",
        "data": {
                        "labels": ["Scream I", "Scream II", "Srceam III"],
                        "datasets": [{
                                        "label": "Voting average",
                                        "data": [7.5, 8.2, 6.7]
                                }]
                }
}
```

404

An error message for indicating there went something wrong.

Webservices

Dit document bevat de design keuzes voor mijn API.

Als eerst zien we de structuur van de API, voor de opdracht zelf heb ik voor de gevraagde onderdelen al zoveel mogelijk proberen rekening te houden met RESTfulness.
De structuur ziet eruit als volgt:

```
/                                    # 1
/movies                              # 2
/movies/{movieid}                    # 3
/movies/{movieid}/same-genres        # 4
/movies/{movieid}/similar-runtime    # 5
/movies/{movieid}/overlapping-actors # 6
/movies/popular                      # 7
/movies/compare                      # 8
```

1, 2

Deze 2 geven standaard 20 films terug van de website.
Deze routes heb ik moeten toevoegen voor er voor te zorgen dat de route 'hackable' is.

3

Deze route geeft de enkel de informatie voor een gevraagde film terug wanneer we een GET request uitvoeren. Deze route was vooral bedoeld voor wanneer we een film willen liken/unliken of deleten. Deze 2 acties worden dan uitgevoerd door een PUT en DELETE.

4, 5, 6

Deze routes worden zo opgesteld dat we een actie kunnen uitvoeren op een film. We filteren dus eerst op een movie id in movies en dan vragen we bijvoorbeeld films op met dezelfde genres. Op deze manier leek het me het duidelijkste, om de route te begrijpen.

7

Deze route geeft de populaire films terug. Dit kan gedaan worden door het argument 'amount' te gebruiken. Als dit argument niet is meegeven dan wordt er standaard de eerste 20 populaire films gegeven. We gebruiken hier een argument in plaats van een parameter omdat we hier meer een query operatie uitvoeren, we krijgen niet meer over minder te zien dan wat popular standaard toont qua informatie, maar een bepaalde scope van die informatie.

De compare route geeft alle informatie terug die gebruikt kan worden om die chart API op te roepen. Ik heb hiervoor gekozen om enkel de data terug te geven dat als argument moet worden gebruikt in plaats van de volledige url. Zodat wanneer de gebruiker toch een andere API hiervoor zou gebruiken, hij enkel de nodige data kan gebruiken uit mijn gegeven data. Anders zit het standaard verbonden aan de url en dat vond ik in het algemeen minder handig.
Hier wordt verder ook een argument 'movies' meegeven waarbij meerdere films hun id kunnen worden meegeven gescheiden met een komma. Als er geen argument wordt meegeven zal er gewoon lege data terug worden gegeven.

Running the project

Er is een run.sh file voorzien. Deze zal een environment aanmaken en alle nodige python package installeren. Nadien zal deze ook automatisch de website openen.