Ostbayerische Technische Hochschule Regensburg
Faculty for Mathematics and Computer Science

IMDB Software of Hollywood Actors and Actresses

Applied Data Science with Python

December 17, 2021

Author: Anna Breithaupt
MatNr. 3202199

Contents

1. Task 1

1.1. Short Description of the Problem

The goal of the project is to create a user-friendly application that provides information about the top 50 actors and actresses.
The information about the movies can be accessed through the IMDb page of the 50 most popular actors (Top 50 Popular Hollywood Actors and Actresses | IMDb, 2013).

1.2. General Approach

To implement this task, I want to scrape the required information from the website, sort it and then store it in a database.

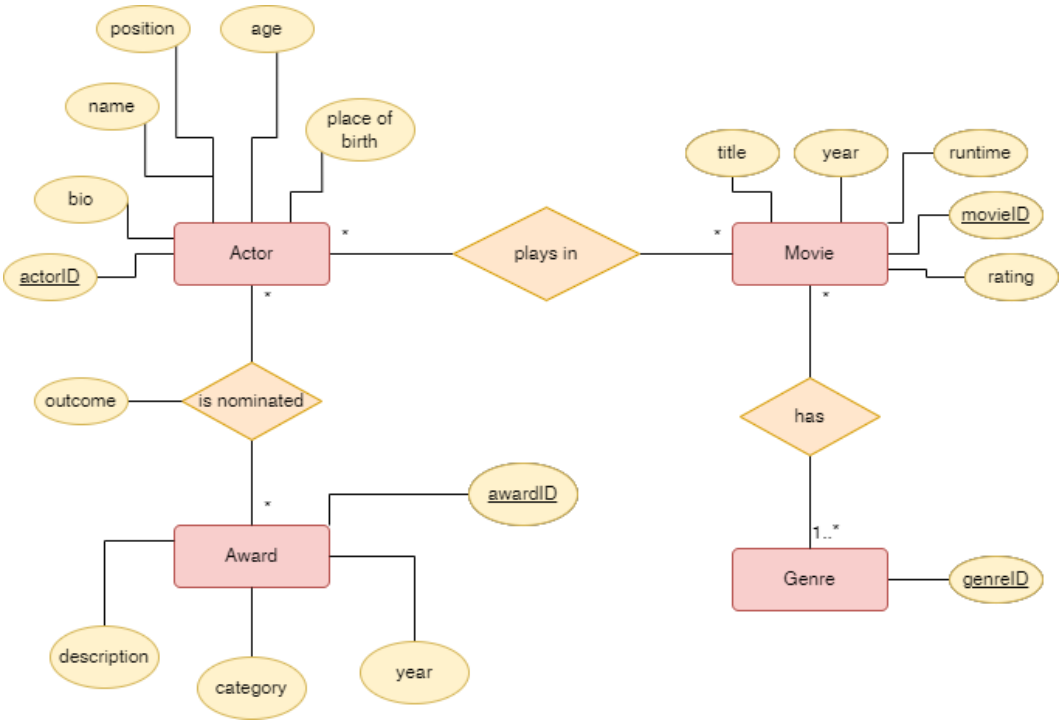


Figure 1: Entity Relationship Sketch

I considered the schema of the database for the beginning as seen in figure 1.
The stored data can then be used to retrieve specific information, statistics as well as to create graphs. The data obtained with this will be displayed to the user in a web application.

1.3. Tools

As development environment I will use **PyCharm**. Besides refactoring and debugging possibilities PyCharm also supports me in the organization and structuring of my project.
As versioning tool I will use **git**, whereby PyCharm also supports versioning with git.

1.3.1. Documentation

For documentation of the code I plan to use **Sphinx**, if more diagrams are needed for planning I will use **draw.io**. Additionally I will use **LaTeX** and **TeXstudio** to write the documentation.

1.3.2. Scraping, Storing and Processing the Data

To get the data from the website I plan to use **Beautiful Soup**. In order to find the right information I will also use the **Google Inspector Tool** provided by Chrome to locate the HTML tags and classes.
The data then will be stored in a database, for which I will use **MySQL**.
To process the data, I will access **Numpy** and **Pandas**. Since I also want to plot, I need **matplotlib.pyplot** as well.

1.3.3. User Interface

For the user to view the data in a descriptive manner, I will use a web interface. For this I utilize the web framework **Django**, **HTML5** and for the graphical elaboration **CSS** and **Bootstrap**.

1.4. Algorithms and Data Structures

1.4.1. Scraping and Storing the Data

The IMDb page lists the top 50 actors (Top 50 Popular Hollywood Actors and Actresses | IMDb, 2013). For each actor a detail page with further information is linked, where query parameters are used. To get the information I need, I thought of the following algorithm. After I have found the first actor, I search for the corresponding query parameter. This will also be used later in my database as ID for the actor. The query takes me to the details page to get all further information about the actor and his awards. On the details page all movies of the actor are listed. The movies are linked again and I can repeat the procedure for the movies. So if the movie is not yet saved, it will be persisted in the database. The movie now can be linked to the actor. The procedure is repeated for all other actors subsequently.
I will mostly use the **find** and **find_all** function of Beautiful Soup.

The information I find with **JSON** formats can be stored in a **dictionary**. The rest of the information can be added to the dictionary and then simply stored in the database using a function. In addition to the lector materials I also used realpython.com as source of information (Beautiful Soup: Build a Web Scraper With Python | Real Python, 2021).

1.4.2. Processing und Evaluation

For the evaluation of the data I would like to use **diagrams**, among other things. To display the awards of the actors I would like to use a **histogram**. So the difference between nominations and wins can be shown well. A **scatter plot** can be used for the average rating of the films in the respective years. The genres of the movies could be shown well in a pie chart or in a **wordcloud mask**. For this I can use different methods from the matplotlib.pyplot.

For returning all actors or movies I will use **lists**, which will then be sorted with a **QuickSort algorithm**.

1.5. Modules

In order to think of suitable modules, I first thought about how to structure the project. For this I roughly fall back on the three layer model, where an application layer, a persistence layer and a presentation layer is implemented. Additionally I was inspired by Hitchhiker's Guide to Python(Structuring Your Project | Hitchhiker's Guide to Python).
In appendix A you can see a rough logic, how I plan to structure the project with corresponding modules.

2. Task 2

2.1. Current state of the project

In the last few weeks I was able to implement large parts of my project. Among other things, I created the databases where the scraped data will be stored. I was able to scrape the necessary information from the IMDb site [unk13] and store it in the database.
Furthermore, I implemeted queries to retrieve the data from the database again and pass it to the web interface. I will explain my exact procedure on the following pages.

2.2. Database creation

2.2.1. Updated entity relationship model

I was able to implement the databases mostly as described in chapter 1.2. On figure 2

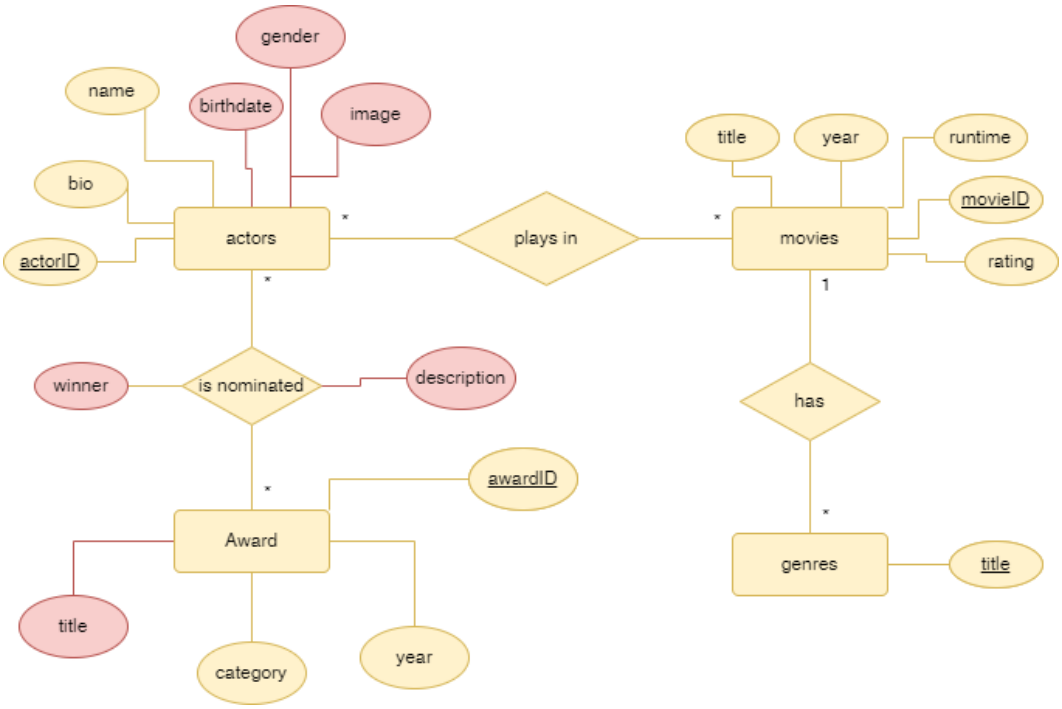


Figure 2: Updated ER diagram

you can see the updated entity relationship model. The changes are highlighted with red color.
Since an actor can be nominated several times for the same award in one year for different roles or categories, I have stored another attribute with the name *description* in the relationship between actor and award. Here, among other things, the nominated role or co-nominated actors can be stored.
I changed the relation between movie and genre from a many to many relation to a one to many relation, because the genre has no attributes except the title. This saves

me an additional table that would reference both.
For the actor, I added image and gender as attributes and changed age to birthdate, which makes it easy to determine the age.

2.2.2. Creation of the database

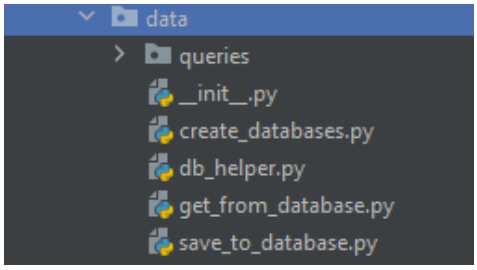


Figure 3: data layer structure

I started to create the database in the data layer of my project. As you can see in figure 3 I created a script called *create_databases.py*. The first thing I want to do is to establish a connection to a MySQL database. Since I will need the connection in the course of my project again and again, I have decided to outsource the function in a separate module *db_helper*. After a connection was established, a new schema will be created, if it does not exist yet. Then a new connection is established, this time directly to the schema.
Then the tables are to be generated. For this purpose, each table is first checked to see if it already exists in the database and dropped if necessary. This way I can avoid errors and make sure that there is no unwanted data in the table. Afterwards the table can be created.

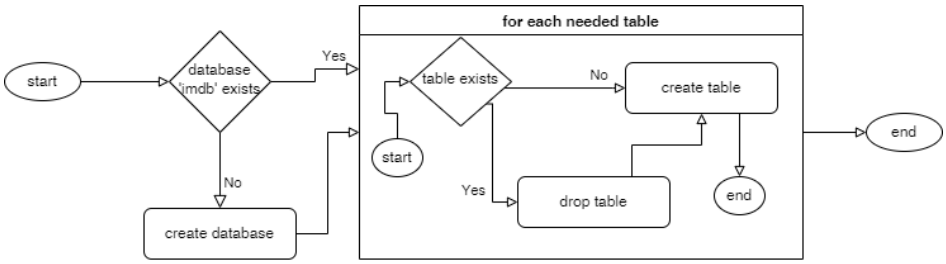


Figure 4: Flowchart create databases

In Figure 4 you can see the described chronology as a flow chart.

2.2.3. Result of the database creation

After the script was called, a database with six tables is created.
The *actors*, *awards* and the *movies* with the *genres* tables contain the main information, the others display the relationships.

	Tables_in_imdb
1	actors
2	actors_have_awards
3	actors_in_movies
4	awards
5	genres
6	movies

Figure 5: created databases

2.3. Scraping of the data

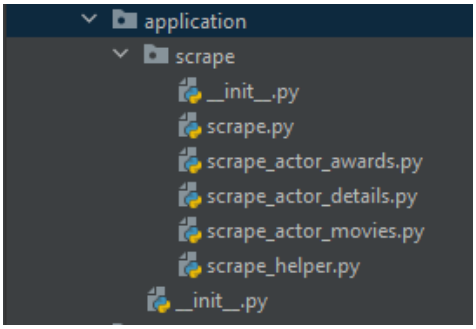
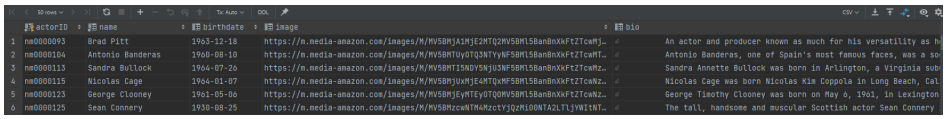


Figure 6: application layer structure

To get the data we want to store in our created database, we switch to the application layer of the project. The script responsible for scraping the data is *scrape.py*. Here we first use BeautifulSoup to fetch a JSON object with all actors. Among other things, the id of each actor is contained here. Via the JSON object I can iterate through all actors in the list. For the exact information I created more scripts, which should give the project clarity. So for each actor you can first scrape the awards and the movies next. In the appendix B, figure 12 you can see a flowchart of the general scraping process in *scrape.py*

2.3.1. Scrape Actor Information

To find out more about the actor, I used the URL *https://www.imdb.com/name/id* and *https://www.imdb.com/name/id/bio* . This turned out to be quite easy. I was able to retrieve some data using the JSON format. The remaining data was found by using html tags. I wrote the data into a dictionary and stored it in the database.
In the following figure 7 you can see some lines of the extracted data.



	actorID	name	birthdate	image	bio
1	nm0000093	Brad Pitt	1963-12-18	https://m.media-amazon.com/images/M/MV5BMjA1MjE2MTQ2MV5BMl5BanBnXkFtZTcwMj…	An actor and producer known as much for his versatility as h
2	nm0000104	Antonio Banderas	1960-08-10	https://m.media-amazon.com/images/M/MV5BMTUyOTQ3NTYyNF5BMl5BanBnXkFtZTcwMT…	Antonio Banderas, one of Spain's most famous faces, was a so
3	nm0000113	Sandra Bullock	1964-07-26	https://m.media-amazon.com/images/M/MV5BMTI5NDY5NjU3NF5BMl5BanBnXkFtZTcwMz…	Sandra Annette Bullock was born in Arlington, a Virginia sub
4	nm0000115	Nicolas Cage	1964-01-07	https://m.media-amazon.com/images/M/MV5BMjUxMjE4MTQxNF5BMl5BanBnXkFtZTcwNz…	Nicolas Cage was born Nicolas Kim Coppola in Long Beach, Cal
5	nm0000123	George Clooney	1961-05-06	https://m.media-amazon.com/images/M/MV5BMjEyMTEyOTQ0MV5BMl5BanBnXkFtZTcwNz…	George Timothy Clooney was born on May 6, 1961, in Lexington
6	nm0000125	Sean Connery	1930-08-25	https://m.media-amazon.com/images/M/MV5BMzcwNTM4MzctYjQzMi00NTA2LTljYWItNT…	The tall, handsome and muscular Scottish actor Sean Connery

Figure 7: actors table

2.3.2. Scrape Actor Awards

Getting the awards was a bit more complicated. For this I used the URL *https://www.imdb.com/name/id/awards*.
The biggest difficulty was to include the awards where the actor was nominated several times for different roles. Since the table was then structured differently, the same attributes, such as year and outcome, remained empty. To get around this, I used the attributes of the predecessor in this case. My procedure can be seen in the following pseudo code.

```
soup = findSoup(url);
awardNames[ ] = soup.findList.findAll(h3);                    ▷ Find titles
awardList = [mockActor];          ▷ Create a mock award for the first entry
for all title in awardNames do
    award = {};
    award['name'] = title;
    award['actorID'] = actorID ▷ is the same for each award, since script gets called
for every actor
    tableRows = title.findNext(table).findAll(tr);
    for all row in tableRows do                    ▷ for every other attribute
        attribute = findAttribute(row);
        if attribute is empty then
            attribute = awardList[-1]['attribute'];          ▷ use previous if empty
        end if
        award['attribute'] = attribute;
    end for
    awardList.append[award];
end for
return awardList[1:];                                  ▷ remove mock award
```

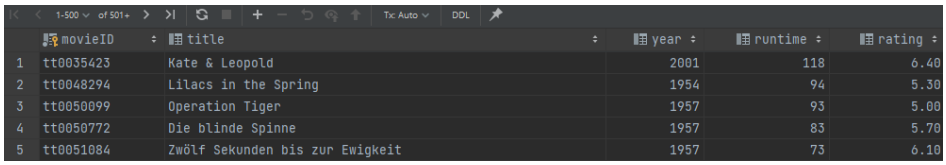
If the award was not already stored in the database by a previous actor, the values had to be stored in the database. Finally, the actor and the award are linked in the relationship table.
The result can be seen in the following excerpt from the database, figure 8

	awardID	year	category	title
1	1	2008	Oscar	Academy Awards, USA
2	2	2005	Oscar	Academy Awards, USA
3	3	2004	Oscar	Academy Awards, USA
4	4	2012	BAFTA Children's Award	BAFTA Awards
5	5	2005	BAFTA Film Award	BAFTA Awards

Figure 8: awards table

2.3.3. Scrape Movies

Scraping the films was done in a similar way as with the actors. The only challenge here was that some attributes were not specified for some films. To avoid NULL values in the database, I stored mock values in the database in this case. Finally the movies and the links to the actors were stored in the database again.
You can see the result in figure 9.



	movieID	title	year	runtime	rating
1	tt0035423	Kate & Leopold	2001	118	6.40
2	tt0048294	Lilacs in the Spring	1954	94	5.30
3	tt0050099	Operation Tiger	1957	93	5.00
4	tt0050772	Die blinde Spinne	1957	83	5.70
5	tt0051084	Zwölf Sekunden bis zur Ewigkeit	1957	73	6.10

Figure 9: movies table

2.4. User Interface

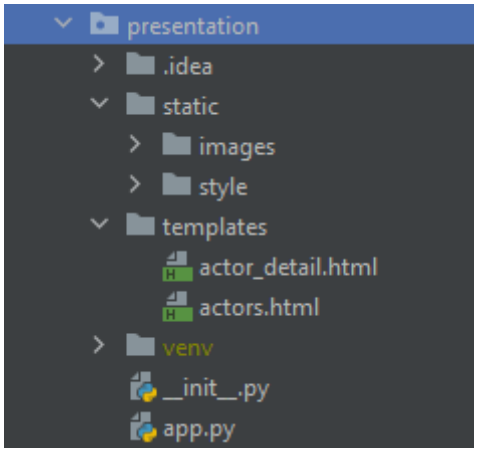


Figure 10: presentation layer structure

The data was scraped from the website and persisted into the database. In the next step I want to display the data in a user-friendly way.
Although I had initially considered using the Django web framework for the implementation of the web interface, I realized during the first implementation attempt that this would not benefit my already existing databases. With Django, databases are created automatically. I preferred to implement it in a way, so the data could be fetched from the database with an already fitting selection and then just pass the needed data to the web interface. Therefore I switched to the web framework Flask. For this I used the Flask documentation [unk] and a tutorial [Dyo21]. In figure 10 you can see the adapted structure. So far I already created a template, on which all actors are displayed as a list. You can see the result so far in figure 11.
To get the required data, I created a module in the application layer that fetches the

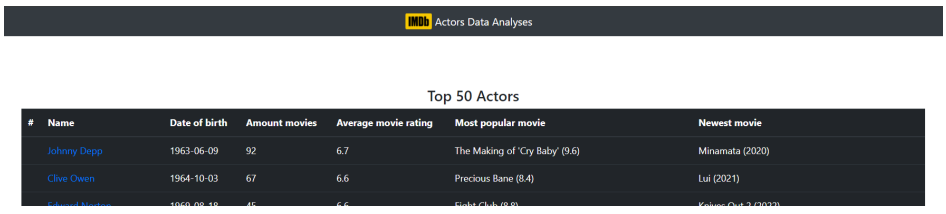


Figure 11: web actors list

data from the database via the persistence layer and returns it in a dictionary.

2.5. Outlook

So far, the data is scraped from the IMDb site, persisted in a database and displayed as a list on a web interface.
The next task is to evaluate and display the data for each actor as well. For this the data must be selected from the database and evaluated in the form of diagrams. In addition, html templates are to be created.

A. Project Structure

```
imdb/
|
├── bin/
|
├── webapplikation/
|      ├──app/
|      |      ├── __init__.py
|      |      ├── admin.py
|      |      ├── apps.py
|      |      |
|      |      ├── migrations/
|      |      |      └── __init__.py
|      |      |
|      |      ├── models.py
|      |      ├── tests.py
|      |      └── views.py
|      |
|      ├──docs/
|      |
|      ├──project/
|      |      ├── __init__.py
|      |      ├── settings.py
|      |      ├── urls.py
|      |      └── wsgi.py
|      |
|      ├── static/
|      |      └── style.css
|      |
|      ├── templates/
|      └──   └── base.html
|
|
├── application/
|      ├── __init__.py
|      ├── runner.py
|      ├── scrape/
|      |      ├── __init__.py
|      |      ├── scrape.py
|      |      └── save_information.py
|      |
|      ├── processinformation/
|      |      ├── __init__.py
|      |      ├── process_information.py
|      |      └── get_information.py
|      |
|      ├── tests/
|      |      ├── scraping_tests.py
|      |      └── information_processing_tests.py
|      |
|      ├── docs/
|      |      ├── scrape.md
|      └──    └── processinformation.md
|
├── data/
|      ├── __init__.py
|      ├── create_databases.py
|      ├── save_to_database.py
|      ├── load_from_database.py
|      ├── docs/
|      |      └── . . .
|      ├── tests/
|      └──    └── . . .
|
├── .gitignore
├── LICENSE
└── README.md
```

B. Flowchart Scraping

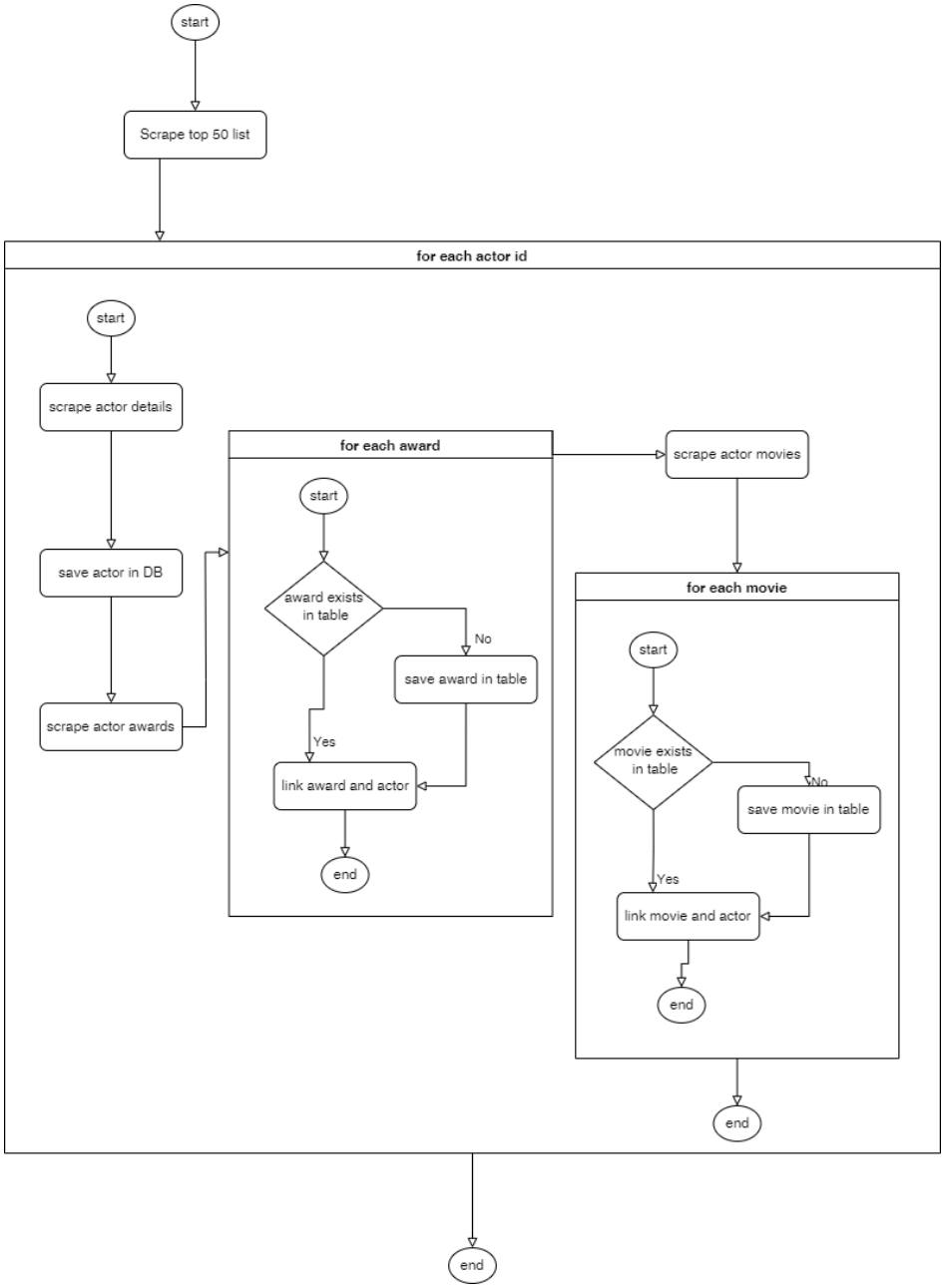


Figure 12: Flowchart scrape.py

References

[Dyo21] Dyouri, Abdelhadi: *How To Make a Web Application Using Flask in Python 3*. https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3. Version: 12 2021

[Rea21a] Real Python: *Beautiful Soup: Build a Web Scraper With Python*. https://realpython.com/beautiful-soup-web-scraper-python/. Version: 11 2021

[Rea21b] Real Python: *Get Started With Django Part 1: Build a Portfolio App*. https://realpython.com/get-started-with-django-1/. Version: 11 2021

[unk] *Welcome to Flask — Flask Documentation (2.0.x)*. https://flask.palletsprojects.com/en/2.0.x/

[unk13] *Top 50 Popular Hollywood Actors and Actresses*. https://www.imdb.com/list/ls053501318/. Version: 05 2013

[unk21] *Structuring Your Project — The Hitchhiker's Guide to Python*. https://docs.python-guide.org/writing/structure/. Version: 2021