What does the application do, what features does it have?

Our application is a webpage that helps people (especially movie creators and directors) to gain important knowledge that can help them in the process of directing a movie, it can be also very useful to other people since it has a lot of information about movies, actors and genres. Some of the features of the app require the user to insert an input and get an output, others are only a button that you click on and it will immediately output the desired data.

Let's say that you are a director of a movie and you want to create a new movie, then you can simply open our app, and use that feature: click on the button that sorts the genres according to the average of budget/revenue of all the films of that genre, this will output for you the genres sorted from the most profitable to the least profitable, then as a director you would like to create a new movie that would be profitable so let's say that you are interested in the top 5 genres according to their profit, now, you can take each one of these genres and enter them in these 3 features that we have:

1) the first feature we can use with these genres is the feature that takes a genre as an input and returns popular actors that have played in the highest number of high rated movies in that genre, which means that they are experienced in movies of that genre and the movies they acted in are high rated and they did a good job, so that means that given a genre name, you can find the most suitable actors for it, which is a great thing for a director!

2) the second feature we can use with these genres is the feature that takes a genre as an input and returns the companies that have created the highest number of movies in that genre, this tells us that these companies are experienced in making movies from that genre therefore you as a director would like to collaborate with them.

3) the third feature we can use with these genres is the feature that takes a genre as an input and returns the top 100 movies in that genre sorted according to their rating, this will tell you as a director what movies succeeded in the genre that you trying to make a movie in, therefore you can take those movies that succeeded and get information about them and then you can do the same things for your movie too, or maybe you could talk with directors of these movies and take some tips from them for making successful movies in that genre.

Another type of features is, let's say that you are a director again, and you have a favorite actor that you would like him to take place in a new movie, so you can use the feature that takes an actor name as an input and it outputs the best genres he acted according to the average ratings of these movies, which means you can know what genres are suitable for your favorite actor and make your new movie to be in these genres.

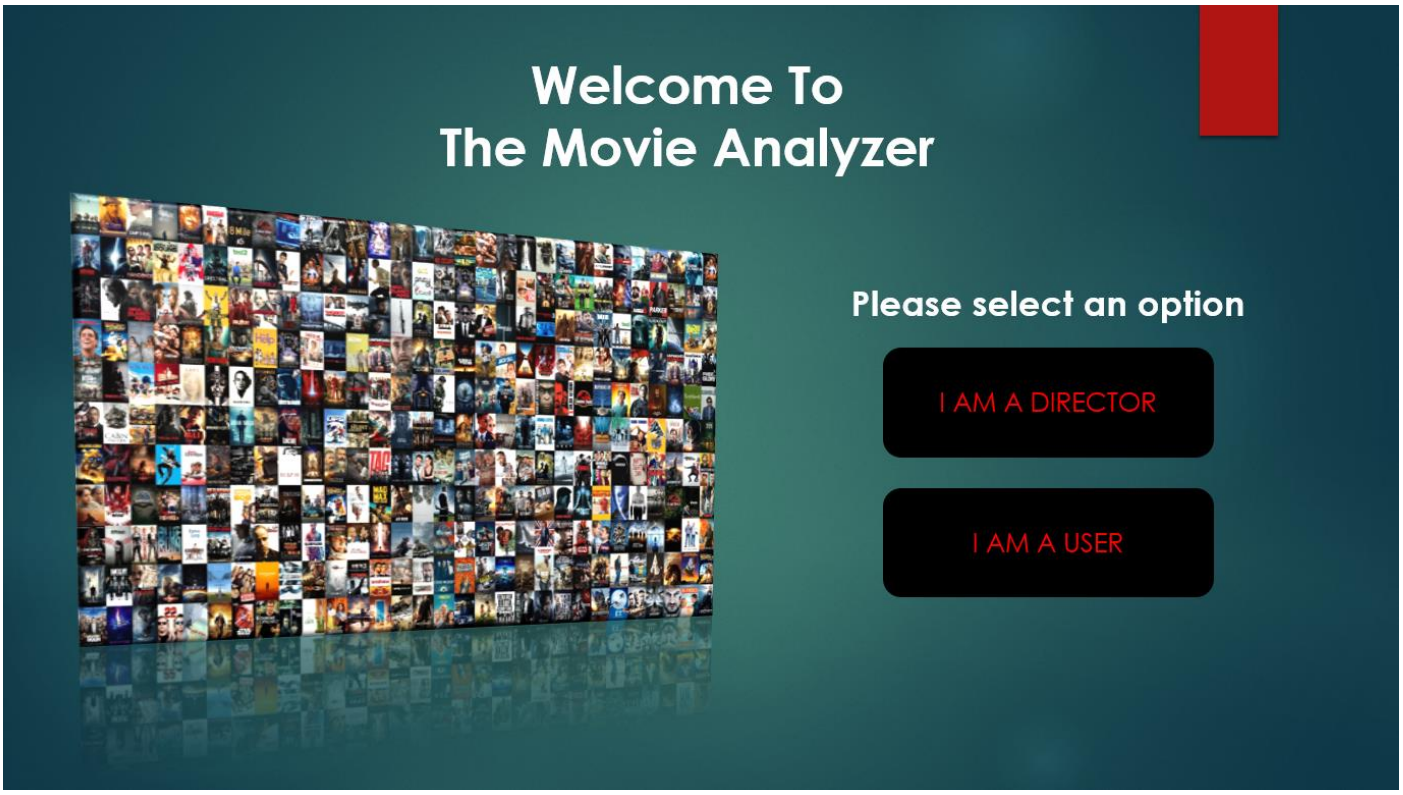
The app is also useful for regular people, let's say that you are a regular person that is bored and you want to watch a movie about a war, so you can use our feature that takes a word as an input, and returns all the movies that have that word in their overview attribute, which means, you will get as an output all the movies that have the word war in their description, and then you can go and watch these movies. Also there is a button that sorts for you the actors according to the number of movies that they acted in, this can widen your knowledge of actors and you can start watching movies of experienced actors. the amount of features that can be used are endless and everybody can find something useful for them in the app.

The design of the application: How it would have looked like if it was implemented?

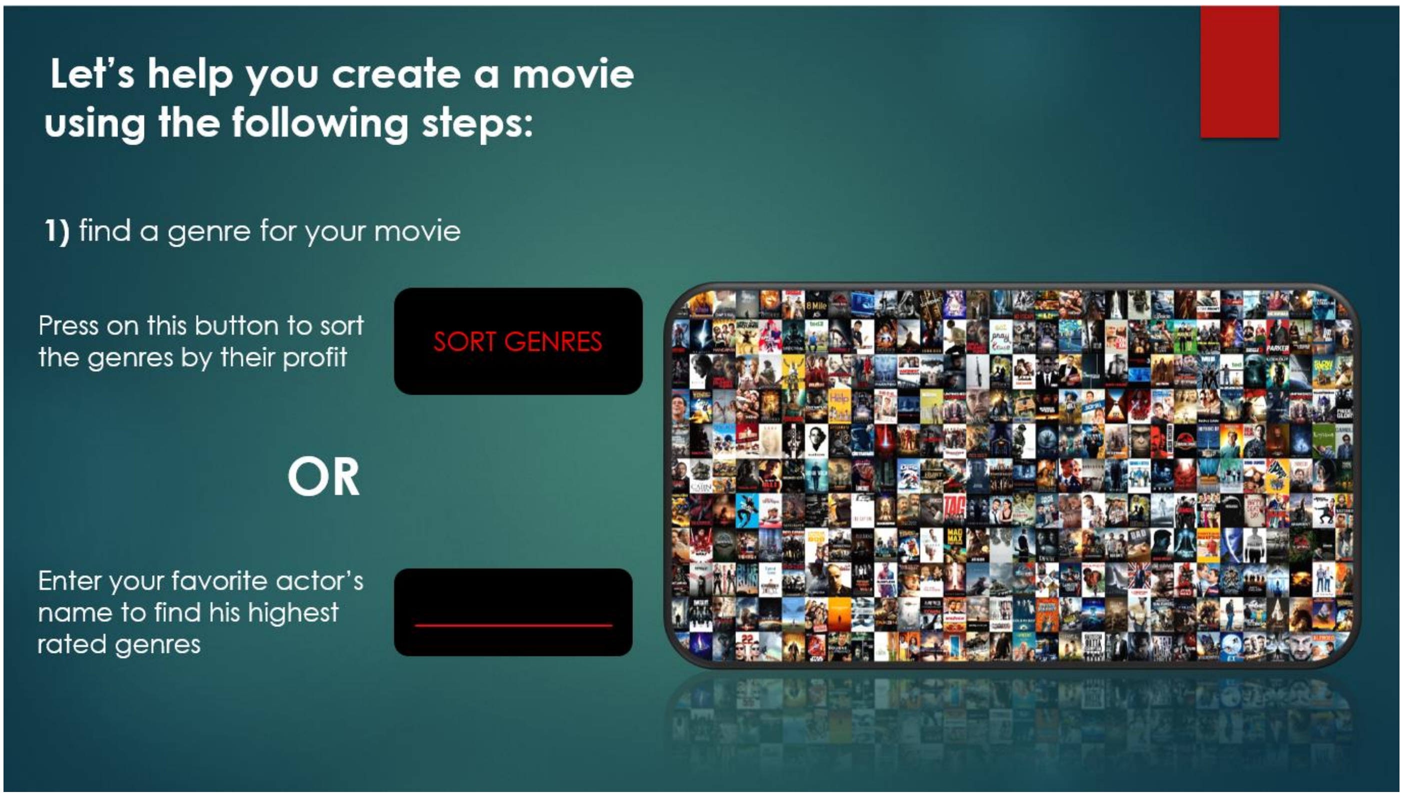
(The queries are numbered relatively to the queries in the python file (QUERIES.py))

If the app was implemented it would have looked like that:

The name of the app is The Movie Analyzer, and this would be the main page of it, there are 2 buttons, the first one is for directors and the second one is for regular users.



If you click on the first button, which means that you are a director looking for creating a movie, then you will be directed to that page:



Let's help you create a movie using the following steps:

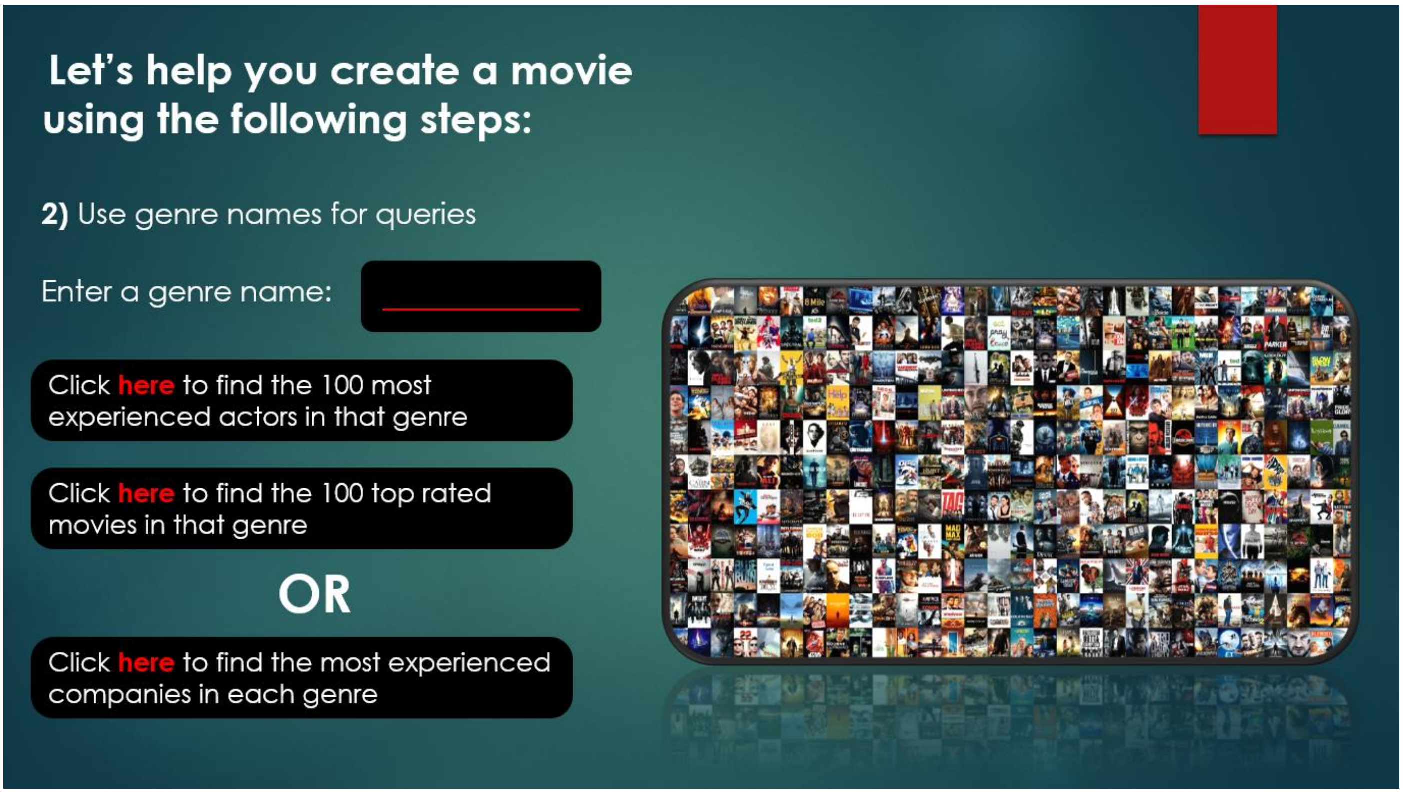
1) find a genre for your movie

Press on this button to sort the genres by their profit

OR

Enter your favorite actor's name to find his highest rated genres

This page will help you decide what genre the movie will be. In order to do that there are 2 options, the first option is to click the button SORT GENRES which will use query 2 from our QUERIES.py file, this query will sort the genres of the movies by the average of the (bedget/revenue) of the movies in that genre, and that can help you decide what genre to pick since the genres are sorted by the profit that they produce, the second option is to enter a name of an actor, and the app will use the 5th query which will sort the genres by the rating of movies that this actor played in. so the client can use these 2 options to decide what genre they want to proceed with, and then they will be directed to the next page:



Let's help you create a movie using the following steps:

2) Use genre names for queries

Enter a genre name:

Click here to find the 100 most experienced actors in that genre

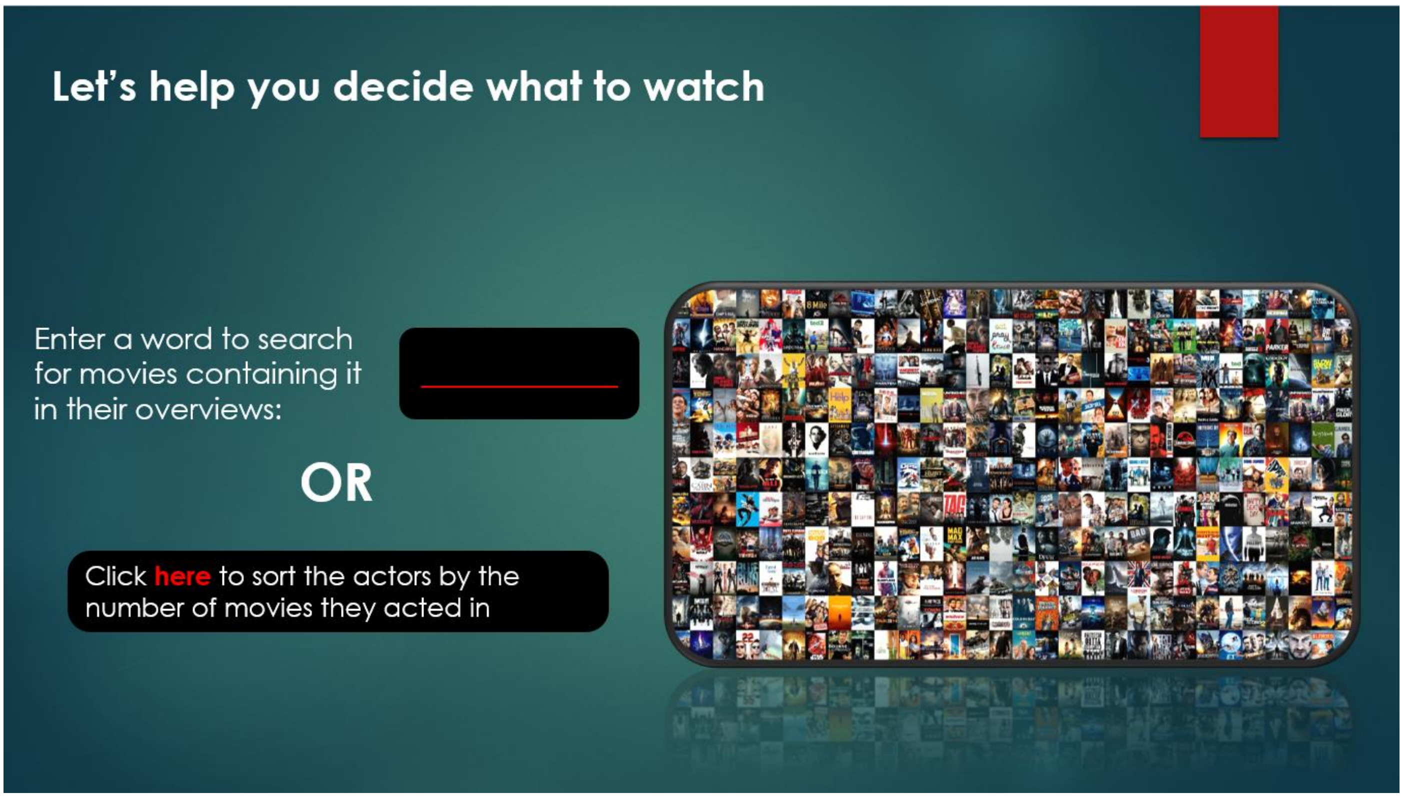
Click here to find the 100 top rated movies in that genre

OR

Click here to find the most experienced companies in each genre

In that page, we can use the genres we found in the previous step to find more information, so the first thing we need to do is enter a name of a genre we want information about, and then there are 2 options, the first option is a button that uses the 7th query, which gives us the top 100 actors in that genre based on the number of movies they acted in, this will help the director choose an actor for his movie, the 2nd option is to click on a button that returns the 100 top rated movies in that genre, by using the 6th query, this will help the director find successful movies in his genre and this can help him learn what to do better. Also there is an option of a button that gives you for every genre the best company for it, for that button we use the 3rd query, this can help the director choose a company to work with on the movie.

If the user picked the second option of I AM A USER in the main page, they will be directed to that page:


Let's help you decide what to watch

Enter a word to search for movies containing it in their overviews:

OR

Click here to sort the actors by the number of movies they acted in

Where they will have 2 options, the first option lets them enter a word, and it will use the 4th query, which will find all the movies that have that word in their overview, this can help people that want to watch a movie about something specific (for example wars) find all the movies about that topic, also there is another option for the user, which is a button that sorts the actors by the number of movies they acted in, this can widen the knowledge of the users and also they can start watching more movies that have experienced actors in them.

Code Structure:

The documentation of our code can be found in the python files, we documented the code there since it is easier to understand the explanations there, here we will talk about the structure of our code briefly

Our code can be divided into 5 parts:

1. This part contains the 7 queries for our app, we made a separate function for each query. After we parse the given input from the user, we call the function that corresponds to the query that he is using and give it as an input the input we took from the user, then this function will execute the sql queries using the cursor object, this code can be found in the QUERIES file (more details on the communication between the client and the server later in the GENERAL FLOW section).
2. The part where we used the API (themoviedb) in order to get data about movies, actors, crew, genres and companies this part is implemented in the API_RETRIEVE file
3. The part where we filtered the data that we got and created intermediate tables that connected those tables to each other, which can be found in the DATA_FILTER file.
4. The part where we create the database which is in the CREATE-DB file.
5. The part where we insert the data to the database which is in the INSERT_DATA_TO_DB file.

Description of the API + how did we use it:

The first step in building our database was finding on the internet a csv file that contains around 10000 movie IDs, we took that file, and choose around 4000 movies from it and saved them in a separate file, then we wanted to create a table in our database that contains all the 4000 movies that we found with data about them, that's why we needed an API that gives us this data, the API we used was:
https://developers.themoviedb.org/3/getting-started/introduction

in that API there are a lot of sections, for this table we entered the movies section, and used the get_details call, which takes a movie_id and returns a lot of data about that movie (all the data in the movies table is from there) in order to do that we need to do this call:

https://api.themoviedb.org/3/movie/{movie_id}?api_key=<<api_key>>

where instead of {movie_id} we put the real movie_id and instead of the <<api_key>> we put our api key that the API gave us, in order to run that for all the 4000 movies that we have we used a python code which is inside of the API_RETRIEVE file.

Now we have the table of the movies stored as a csv file, but for every movie we have a list of genres of that movie (this is part of the data that the API returns for us), since a table in our database can't contain more than a single value in an entry, we needed to create a separate table for it (genres), in order to do that, we used the section of genres in the API and used that call:
https://api.themoviedb.org/3/genre/movie/list?api_key=<<api_key>>

this call returns for us all the genres that are available for movies and their IDs, and from them we create a csv file that contains the genres and their IDs. There are 19 genres in total.

The API returned for us a list of companies that created each movie in the movies table, in the API_RETRIEVE file we wrote a code that collects the IDs of all the companies of the movies, and then uses the companies section in the API to collect data about the companies, we used the following call:

https://api.themoviedb.org/3/company/{company_id}?api_key=<<api_key>>

where we replace the {company_id} with our company_id, we do that using the python code in API_RETRIEVE and we create a companies table.

For getting the actors and crew we used the get credits call which is inside the movies section in the API, this call takes a movie id and returns data about its cast and crew, in order to do that for all the movies we used a python code which is in the API_RETRIEVE file. This is the call for getting the credits:

https://api.themoviedb.org/3/movie/{movie_id}/credits?api_key=<<api_key>>

where we replace the {movie_id} with our movie_id. After that call we will have a table that for each movie it contains a list of actors and a list of crew that were a member of that movie. In order to get information about those actors and crew members we use the get_details call in the people section in the API, this call takes a person id and returns relevant data about that person, and that is how we take the credits csv file and make two csv files from it, one for crew and the other for actors, we do that using the code in DATA_FILTER file.

Using this API we built the 5 main tables in our database: movies, actors, crew, genres, companies. Now because for every movie there can be more than one actor and every actor can be a member of more than one movie we need to create a table called movie_actor which will contain IDs of movies and actors where the actor participated in the movie, this table will have foreign keys (the ids) for both the movies and actors tables. The same concept applies for the other tables, therefore we create the tables: movie_crew, movie_genre and movie_company (more information about that in the SOFTWARE-DOCS file) in order to create these tables, we needed to write a python code that goes over the lists in the movies table and for every movie it inserts the relevant data to the relevant table (this code is in the DATA_FILTER file), for example if the movie with id 123 has a list of genres[{1:comedy}, {2:horror},{3:animation}] then we insert to the movie_genre table the following rows: (1,1) , (1,2) , (1,3) because that movie has these three genres. This concept works for all the other tables with the movies table. And that is how we used the API and built our csv files that contain our data, then we created the database and inserted to it this data using the python code in the SRC folder.

Why did we choose this DB design?

We wanted to keep things simple but on the other hand we wanted to make it interesting, therefore we choose to have a table called movies in the middle, and all the other table will have relations only with the movies table for, to keep it interesting we wanted to have tables that contain useful data, so we choose to have tables for actors, crew, genres and companies since these topics are very important for any regular person that wants to watch movies and also for actors and directors of movies in general. The reason we came up with this design is that we first thought about the queries that we wanted to make and then design the database, and using this design we can implement all the queries that we wanted.

Why is this design the most efficient for our needs?

Because some of the tables that we made are very small (for example the genre table contains 19 rows only) therefore queries on that table will run faster, we also made sure that in order to join any two tables the number of joins would be less than 5 (we can see that using the diagram in the SOFTWARE-DOCS file) and that can make our program run faster since there are less joins, also this design contains an index on a text column which improves the runtime of the fulltext query that we have.

What are disadvantages of other designs we have considered?

We considered adding another table called reviews, which will contain reviewers and movies and a rating for the movie and a brief explanation of why the reviewer gave that rating, but we thought that it would be better to use the rating attribute in the movies table since it averages all the ratings of the reviewers of that movie, and also it saves us from creating 2 more tables which could take a lot of space. We have also considered to union the crew and actors tables since they have the same data, but we didn’t because then we will need to store for every person another column that indicates if the person is an actor or crew member and also because an actor has a role in the movie but the crew member has a job so we needed to separate them.

General flow of the app:

When the user clicks a button to request a certain type of data:

On the client side:
The browser sends a GET HTTP request to our server with the arguments:

- htq = the number of the relevant type of data he asks for (we numerate the requests in the html file relatively to our "QUERIES.py" file)
- htarg = the argument needed to return the data (genre_name/actor_name/word)

On the server side:
The server has a listener to any HTTP GET requests, it receives the request sent by the user and in the request it has the values htq=i?htarg=j, it checks that i and j are legal, or in other words, it checks that 1<=i<=7 and j is actually a string that contains (genre_name/actor_name/word) with respect to i**.**

if i and j aren't legal it sends "Bad http request" back to the user as a response, otherwise the server executes query I, with the argument j if needed, then it wraps the results into a http response and sends it back to the user

On the client side:
The browser receives the response from the server, fetches the response and prints it to the user in a well-designed format and in the right place according to the html code of the webpage.