Wednesday: Request Object

Search for Movies

We are going to implement a search functionality from our API. Let us first create the search request.

request.py

```
def search_movie(movie_name):
    search_movie_url = 'https://api.themoviedb.org/3/search/movie?api_key={}&query={}'.format
(api_key,movie_name)
    with urllib.request.urlopen(search_movie_url) as url:
        search_movie_data = url.read()
        search_movie_response = json.loads(search_movie_data)

        search_movie_results = None

        if search_movie_response['results']:
            search_movie_list = search_movie_response['results']
            search_movie_results = process_results(search_movie_list)


    return search_movie_results
```

We use a new URL for our search request that passes in our API key and the movie name then we create the request and process the results. We then create the view function for the for the search route.

views.py

```
from .requests import get_movies,get_movie,search_movie
```

We then import the `search_movie()` function from the *requests.py* file.

We then create the search view function that will display our search items from the API.

views.py

```
...
@app.route('/search/<movie_name>')
def search(movie_name):
    '''
    View function to display the search results
    '''
    movie_name_list = movie_name.split(" ")
    movie_name_format = "+".join(movie_name_list)
    searched_movies = search_movie(movie_name_format)
    title = f'search results for {movie_name}'
    return render_template('search.html',movies = searched_movies)
```

We create a `search` view function that has passes in a dynamic variable. We then format the `movie_name` to add `+` sign between the multiple words.

We then call the `search_movie()` and pass in the formated movie name. We then pass the `searched_movies` list to our template.

Let us create the search template.

search.html

```
{% extends 'base.html'%}
{% import 'macros.html' as macro%}

{% block content %}
    <div class="container">

        <div class="row">

            <h3> Found {{ movies| count }} matches </h3>
            <hr>
            <ul>
                {{ macro.displayMovieList(movies)}}
            </ul>

        </div>

    </div>
{% endblock %}
```

We extend the *base.html* file then we import the *macros.html* file.We use the `count` filter that counts all the items in a list. Then we call the `displayMovieList()` macro that will display all the movies.

Let us then create the search form in the *index.html* file.

index.html

```
...
{% block content %}

<div class="container">
    <form>
        <input type="text" name="movie_query" placeholder="Search for a Movie" class="form-co
ntrol">
        <input type="submit"  value="Submit" class="btn btn-default">
    </form>

</div>
...
```

Then create a form with two Input fields. One to get a text input with the name `movie_query` the other to adds a submit button.

Request Object

The request object is provided by flask and it encapsulates our HTTP request with all its arguments to the view function.

views.py

```
from flask import render_template,request,redirect,url_for
```

We first import the `request` object from flask inside the the *views.py*. We will also import the `redirect` function and `url_for` functions which we will see what they do later on.

views.py

```
# Views
@app.route('/')
def index():

    '''
    View root page function that returns the index page and its data
    '''

    # Getting popular movie
    popular_movies = get_movies('popular')
    upcoming_movie = get_movies('upcoming')
    now_showing_movie = get_movies('now_playing')

    title = 'Home - Welcome to The best Movie Review Website Online'

    search_movie = request.args.get('movie_query')

    if search_movie:
        return redirect(url_for('search',movie_name=search_movie))
    else:
        return render_template('index.html', title = title, popular = popular_movies, upcomin
g = upcoming_movie, now_showing = now_showing_movie )
```

We update our index view function. When we submit the form inside our *index.html* it creates a query with the name of the input `movie_query` and the value as the input value. We get the query in our view function using `request.args.get()`function. We pass in the name of the query and the value is returned.

We then check if the value actually exists if it does we use the `redirect` function that redirects us to another view function. We then pass in the `url_for` function that passes in the `search` view function together with the dynamic `movie_name`which assign it to our form Input value.

We can run our application and see our form. We can input a movies name and see results be loaded.

You can find the application at this point here https://github.com/mbuthiya/watchlist/tree/12-Form-Requests

Creating A review Class

Now that we can successfully display movies from our API and our search functionality works perfectly, let's take it a step further and allow users to give reviews for movies they like.

Let us first create the Reviews class inside our *models* folder.

models/reviews.py

```
class Review:

    all_reviews = []

    def __init__(self,movie_id,title,imageurl,review):
        self.movie_id = movie_id
        self.title = title
        self.imageurl = imageurl
        self.review = review


    def save_review(self):
        Review.all_reviews.append(self)


    @classmethod
    def clear_reviews(cls):
        Review.all_reviews.clear()
```

We create a `Review` class that has a `__init__()` method that takes in the Movie ID, The review title, The image URL and the review itself. We then have a `save_review`method that appends the review object to a class variable `all_reviews` that is an empty list. We then have a `clear_reviews` class method that clears all the Items from the list.

You can find the application at this point here https://github.com/mbuthiya/watchlist/tree/13-create-movie-review-class