Tuesday: Links and Static Files; Error Pages

Links and CSS

We can now create our movie details page in our application. First we need to get the movie.

request.py

```
def get_movie(id):
    get_movie_details_url = base_url.format(id,api_key)

    with urllib.request.urlopen(get_movie_details_url) as url:
        movie_details_data = url.read()
        movie_details_response = json.loads(movie_details_data)

        movie_object = None
        if movie_details_response:
            id = movie_details_response.get('id')
            title = movie_details_response.get('original_title')
            overview = movie_details_response.get('overview')
            poster = movie_details_response.get('poster_path')
            vote_average = movie_details_response.get('vote_average')
            vote_count = movie_details_response.get('vote_count')

            movie_object = Movie(id,title,overview,poster,vote_average,vote_count)

    return movie_object
```

We create a `get_movie()` function that takes in a movie id and returns a movie object. We create a `get_movie_details` URL by formatting the base URL with the `id` and API key.

We then create a request and load the data and create a movie object.

views.py

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

We then import `get_movie` function to our _views.py_ file.

```
@app.route('/movie/<int:id>')
def movie(id):

    '''
    View movie page function that returns the movie details page and its data
    '''
    movie = get_movie(id)
    title = f'{movie.title}'

    return render_template('movie.html',title = title,movie = movie)
```

We then update our movie details route. We create a movie object by calling the `get_movie()` function and pass in the dynamic URL id . We then pass that route into our template file.

movie.html

```
{% extends 'base.html'%}
<!-- Content block -->
{% block content %}
<div class="container">

    <!-- Poster background -->
    <div class="row">
        <div class="col-xs-6 col-sm-6 col-md-6 col-lg-6 posterPath" style="background: url
({{movie.poster}}) no-repeat center center">
        </div>

        <div class="col-xs-6 col-sm-6 col-md-6 col-lg-6 movie-details">
            <h3>{{ movie.title }}</h3>

            <p class="overview"> {{ movie.overview }}</p>
            <p class="ratings"> <b> {{ movie.vote_average }}</b> - <i>{{ movie.vote_count}} v
otes </i> </p>


        </div>
    </div>

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

We create a template with two columns passing in the movie poster on left column and movie details on the right. Now we need to link to this dynamic page.

macros.html

```
{% macro displayMovieList(movie_list) %}

    {% for movie in movie_list %}
        <div class="col-xs-12 col-sm-4 col-md-2 col-lg-2 movie-card">
            <img src="{{ movie.poster }}" alt="{{ movie.title }} poster" class="img-responsiv
e">
            <li class="text-center">
                <a href="/movie/{{movie.id}}">
                    {{ movie.title|truncate(10)}}</a>
            </li>
        </div>

    {% endfor %}

{% endmacro %}
```

We now want to update our `displayMovieList` macro. We add the Movie poster to be displayed . We then add an anchor tag that links to the dynamic URL `/movie/{{movie.id}}`.

We also change the movie title variable block by adding the truncate **filter** . A filter is passed into a variable block to modify the content of of the block and are added after the variable with a pipe `|` character.

The truncate filter shortens a variable size according to the characters we add to its parameters.

Find all the Jinja Built In Features here (http://jinja.pocoo.org/docs/2.9/templates/#list-of-builtin-filters)

Now when we load our application we see each movie as with a movie image and a link. And when we click the link we are taken to the movies details page.

`url_for()`

Flask provides the `url_for()` helper function that generates a URL from information stored in the app URL map. We can use it to link our application to CSS.

Static files like Images,CSS and JavaScript files are given a special route by flask in which they can be accessed `/static/<filename>`

We first create a *css* folder inside our *static* sub folder. We then create a CSS file for our index page. *index.css*

index.html

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

<!-- Styles block -->

{% block styles%}
    {{ super() }}
    <link rel="stylesheet" href="{{url_for('static',filename='css/index.css')}}">
{% endblock %}
<!-- Content block -->
...
```

We use the `block styles` block provided by the *bootstrap/base.html* file. Before we add our content we first create a variable block and call the `super()` function. This tells Jinja not to override any code that is defined in the block. Remember the*bootstrap/base.html* has defined links to the bootstrap css files and we do not want to override those links.

We use the `url_for()` function to create a link to the static file. The function takes in the view function as the first argument, which is `static` and then we pass in the dynamic filename `css/index.css`

We can now manipulate our template file and add styling to it.

You can find the application at this point from here https://github.com/mbuthiya/watchlist/tree/10-Adding-Css

Error Pages

If by some chance a user enters a wrong URL in the browser. Let's say they pass in movies instead of movie. `http://127.0.0.1:5000/movies/396422` they will be greeted with the default `404` page provided by flask. But this page is boring. We can create our own error page and customize it.

Lets add that code to our *app/error.py* file.

app/error.py

```
from flask import render_template
from app import app

@app.errorhandler(404)
```

```
def four_Ow_four(error):
    '''
    Function to render the 404 error page
    '''
    return render_template('fourOwfour.html'),404
```

We import the `render_template()` function and the flask application instance.

We create a new decorator `app.errorhandler()` that passes in the error we receive. We create a view function. that returns *fourOwfour.html* file and we also pass in the status code we receive `404` Now we can go to and create our template file.

fourOwfour.html

```
 {% extends 'base.html'%}

{% block content %}
    <h1>WHOOOPS we can't find that page</h1>
{% endblock %}
```

Lastly, we need to import `error.py` file in the `__init__.py` file.

__init__.py

```
....
from app import views
from app import error
```

You can find the application at this point from here https://github.com/mbuthiya/watchlist/tree/11-Error-Messages