The Movie Database

User Documentation

Kory Prince

Contents

1. Introduction

The goal with this project was to create a very usable web frontend to a movie database. To make the project interesting, a lot of data was used, with the idea being that as much as possible it be based on fact. Movie data (on 860 movies) was pulled from The Movie Database (TMDB). Reviewers were generated using List of Random Names. All ratings are based around the real user rating for each movie (as given by TMDB). In some cases there was no rating in which cases ratings defaulted to 0. All ratings are (re)generated when the application starts.

All data (except for the movie poster images which are stored on TMDB's servers) is stored in a SQLite3 database. A Python uWSGI application running behind nginx serves as a Web Frontend to the data. This application written entirely from scratch, uses custom templates and url mapping system to serve dynamic content to the end user.

The interface is fully Responsive and Progressive. This means that the website can be viewed on older or more restrictive browsers and still work. The site will use features of the browser if present to enhance the experience (Use Firefox or Chrome.) Supports all major browsers (only IE8+. Please, let old IEs die a peaceful death.) Finally, the site scales down on mobile web browsers without sacrificing the content.

All code is given to the Public Domain where possible. TMDB data belongs to that organization.

2. Installing the Application

The Movie Database is a Python application running on uWSGI. Nginx is the webserver we use to interface with a user's browser. Ubuntu Linux is the server OS. This section will guide you on installing each of these components.

2.1. Ubuntu

This guide will not cover how to install Ubuntu Server, but it is fairly straightforward, and many guides can be found on the internet. We used Ubuntu 12.04. You can use this OS that supports python 2.7 and uWSGI.

2.2. The Movie Database Code

The source of The Movie Database is hosted on Github. You can download your own copy by using the following commands:

```
user@server:~$ sudo apt-get install unzip #needed for later command
user@server:~$ cd /tmp
user@server:/tmp$ wget https://github.com/korylprince/schoolprojects/archive/master.zip
user@server:/tmp$ unzip master.zip
user@server:/tmp$ sudo cp -R /tmp/schoolprojects-master/moviedb /opt/
```

This will install the software in the /opt directory of your server.

2.3. Movie Data

The movie data must be generated as redistribution of the data is not permitted. You must obtain an API key from TMDB. Once the key is obtained you can edit the scrape.py file and use it to generate the data.

```
user@server:~$ cd /tmp/schoolprojects-master/moviedb/gen
user@server:/tmp/schoolprojects-master/moviedb/gen$ vim scrape.py #insert your api key!
```

Next you must obtain the pytmdb3 library:

```
user@server:/tmp/schoolprojects-master/moviedb/gen$ wget \
> https://github.com/wagnerrp/pytmdb3/archive/master.zip
user@server:/tmp/schoolprojects-master/moviedb/gen$ unzip master.zip
user@server:/tmp/schoolprojects-master/moviedb/gen$ mv pytmdb3-master/tmdb3/ ./
```

Finally we can run the scrape.py file:

```
user@server:/tmp/schoolprojects-master/moviedb/gen$ python scrape.py
```

This last command can take a very long time because it will pull data from TMDB's servers. It can also consume large amounts of memory. Once done it will produce a movies file in the current directory. You will need to copy this to the application directory:

```
user@server:/tmp/schoolprojects-master/moviedb/gen$ sudo cp movies /opt/moviedb/gen/
```

Finally, we must change the permissions of the application to the web server user:

```
user@server:~$ sudo chown -R www-data:www-data /opt/moviedb
```

2.4. uWSGI

uWSGI can be installed from Ubuntu's package manager using terminal:

```
user@server:~$ sudo apt-get install uwsgi uwsgi-plugin-python
```

Now we must create a configuration for our application:

```
user@server:~$ sudo vim /etc/uwsgi/apps-available/moviedb.ini
```

In this file paste the following:

```
[uwsgi]
pythonpath = /opt/moviedb
chdir = /opt/moviedb
module = web
processes = 1
```

Now we must enable the application and restart the service:

```
user@server:~$ sudo ln /etc/uwsgi/apps-available/moviedb.ini \
> /etc/uwsgi/apps-enabled/moviedb.ini
user@server:~$ sudo service uwsgi restart
```

2.5. Nginx

Nginx is the webserver that will sit between the user's browser and uWSGI. You can use other web servers, but Nginx is very efficient and easy to configure. First to install nginx:

```
user@server:~$ sudo apt-get install nginx
```

Now we must configure it:

```
user@server:~$ sudo vim /etc/nginx/sites-available/default
```

Paste the following code:

```
server {
    listen 80;
    server_name movies.unstac.tk;
    root /opt/moviedb;

    charset utf-8;

    // robots.txt for no indexing
    location /robots.txt {
        try_files /static/robots.txt =404;
    }

    location /static {
        try_files $uri $uri/ =404;
    }

    location / {
        uwsgi_pass unix:///run/uwsgi/app/moviedb/socket;
        include uwsgi_params;
    }

    location ~ (.*.py|.*.pyc|.*.git.*) {
        return 404;
    }
}
```

Of course you must change movies.unstac.tk to your own server name.
Now just restart the server:

```
user@server:~$ sudo service nginx restart
```

You should now be able to browse to your website and view it.

3. Using the Application

You can view a live application at movies.unstac.tk if you don't wish to install your own application.

3.1. Home

Upon visiting the web page, you will be presented with a page much like Figure 3.1. This page contains a Search box and the information found in the Introduction of this document.

Figure 3.1.: Home page

3.1.1. Navigation Bar

The links along the top are present throughout the application. Clicking on Home or clicking the logo will take you to the current index page. Clicking Best, Worst, Random, or Not Rated will bring you to those respective pages.

3.2. Best and Worst

Clicking on Best or Worst will show a page much like Figure 3.2. The 10 Best or Worst movies will be shown.

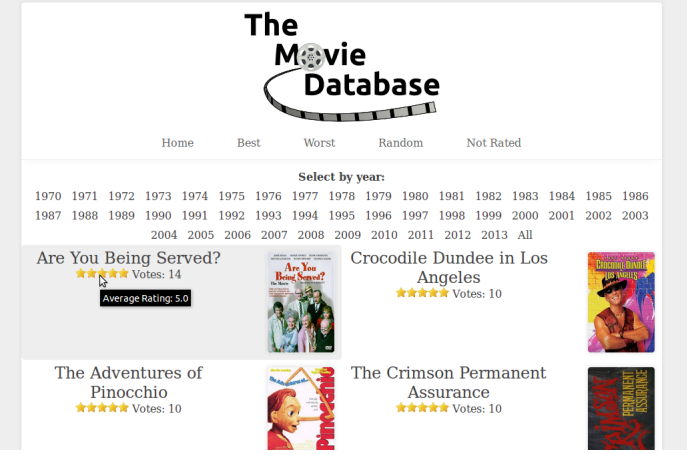

Figure 3.2.: Best page

3.2.1. Year List

You will see directly below the Navigation Bar there is a list of years. Each year that a movie in the database was produced in is listed here. Clicking on the year will bring you to the list of the 10 Best of Worst movies for that year. When showing a year, the year is listed like in Figure 3.3.

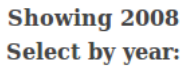

Figure 3.3.: Showing the selected year

3.2.2. Movie List

You will see that for each movie, the title, star rating, and movie poster will be shown. As shown in Figure 3.2, hovering over the stars will give a more exact average. Clicking on either the title of a movie will bring you to that movie's Movie Page.

3.3. Random

The Random page looks much like the Best or Worse page. The only differences are that each time the page is loaded, 10 different movies will appear, and the Year List will does not show on this page.

3.4. Not Rated

The Not Rated page will show any movies that have no ratings. This is the first page you might notice something different at the bottom of the screen, the Pager.

3.5. Pager

As you can see in Figure 3.4, there is a Next → link. This link will take you to the next page of results. This will only show if there are more results than showing on the current page.

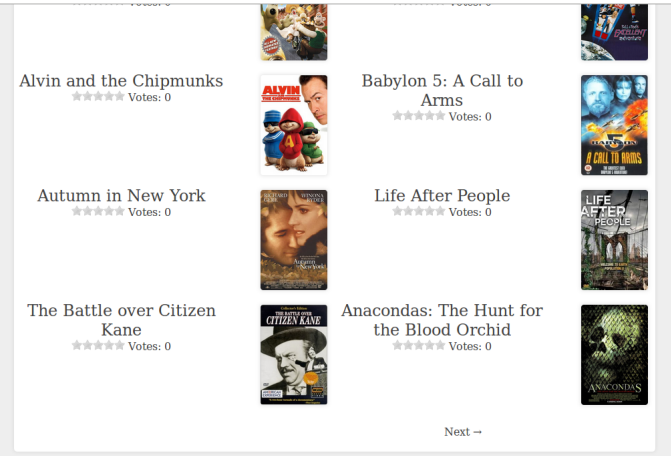

Figure 3.4.: The Next → link

If there are several pages you might see both the Next → link and the ← Previous link as seen in Figure 3.5.

Figure 3.5.: The full Pager

3.6. Search

As seen in Figure 3.1, the Home page contains a search box and button. Simply enter part or all of a movie title and click the Search button as shown in Figure 3.6.

Figure 3.6.: The Search box

If your query returns any results, they will be displayed in the same familiar list. If your search returns no results then you will see the page in Figure 3.7. Simply click the Home button or the link below to try another search.

Figure 3.7.: No results found

3.7. Movie Page

If you click on the poster or title of any movie in a Movie List, you will be taken to the Movie Page. As you can see in Figure 3.8, the Movie Page displays more information about the movie: The title, director, release year, average rating, a summary, and a larger version of the poster. Below this you can see some of the reviewers and the ratings they gave this movie. Clicking on the reviewer's avatar or name will bring you to that reviewer's Reviewer Page.

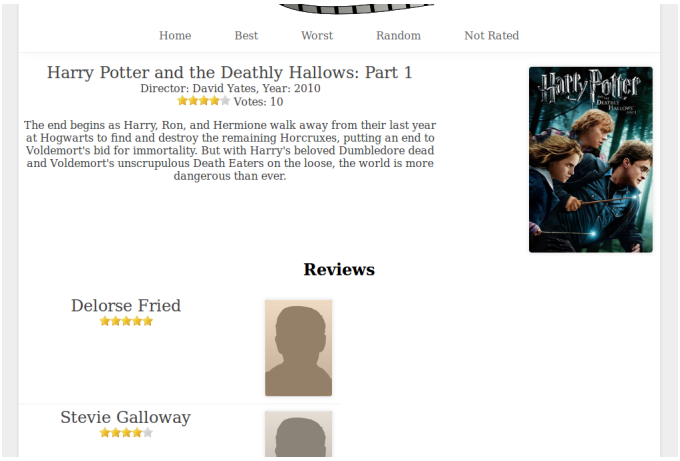

Figure 3.8.: Movie Page

At the bottom of the screen you will see the Click to show all Reviews link as in Figure 3.9. Clicking on this link will show all reviews for the movie.

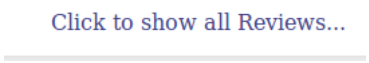

Figure 3.9.: Show all Reviews

3.8. Reviewer Page

Clicking on a reviewer's avatar or name will bring you to their Reviewer Page as show in Figure 3.10. This page is similar to the Movie Page. It will show the reviewer's avatar, average rating given, and number of votes. Below this is a list of movies reviewed by the reviewer and another Click to show all Reviews link that will show all movies reviewed by the reviewer.

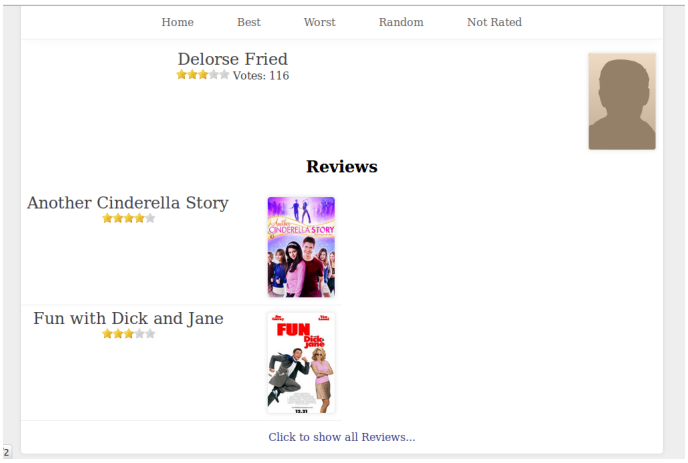

Figure 3.10.: Reviewer Page

4. Mobile Interface

There is no separate Mobile Interface. Rather the application will transform its content to make better use of a smaller screen size. The application uses CSS media queries to scale the content to size, and some Javascript code to make some elements more usable on a smaller interface. If you would like to try out the Mobile Interface and see the content scale down in real time, you can use Firefox's Responsive Design View which comes with recent versions. Simply press CTRL+Shift+M to activate this feature. You can drag the corner handle of the web page to scale the page to any size and see how the application responds in real-time.

4.1. Menu

The Menu will collapse down to a single button on smaller screens. Clicking on the Menu button will show the Navigation links as seen in Figure 4.1.

Figure 4.1.: Mobile Menu

4.2. Year List

The Year List is far too big to show on smaller screens, so it is collapsed into a select box as shown in Figure 4.2. Selecting any year will bring you to that year's page.

Figure 4.2.: Mobile Year List

4.3. Movie List

As seen in Figure 4.2, the Movie list is now only one movie wide, and has collapsed down to better facilitate smaller screens.

4.4. Movie and Reviewer Page

As seen in Figure 4.3, the Movie and Reviewer pages also collapse down to fit better on smaller screens.

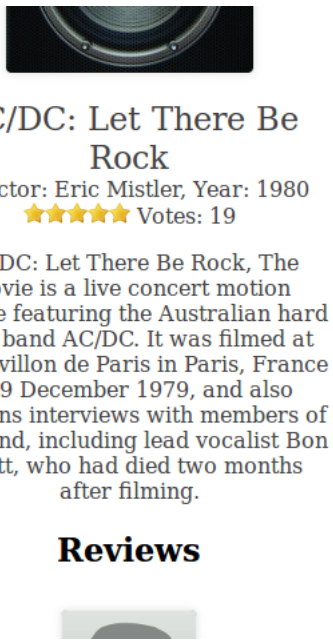

Figure 4.3.: Mobile Movie Page

A. Source Code

The following is the source code for all database related actions. Note that some syntax is SQLite3 specific.

A.1. create.sql

```
create table movie (
    movieID integer primary key autoincrement,
    title varchar(100),
    director varchar(50),
    year decimal(4,0),
    poster varchar(100), /* Extra field for movie poster */
    overview text /* Extra Field for movie description */
);

create table reviewer (
    reviewerID integer primary key autoincrement,
    name varchar(50)
);

create table rating (
    movieID integer,
    reviewerID integer,
    ratingDate date,
    stars integer,
    primary key (movieID,reviewerID), /* While the relation is many to many,
        no reviewer will review a movie twice. Besides,
        I'm generating the data so I know this holds. */
    foreign key (movieID) references movie (movieID),
    foreign key (reviewerID) references reviewer (reviewerID)
);
```

A.2. drop.sql

```
drop table rating;
drop table movie;
drop table reviewer;
```

A.3. db.py

```
import sqlite3
import pickle
import random
import datetime


def create(conn): #expects sqlite3 connection

    # get movie data generated with scrape.py
    # This data is real
    # has (title,director,year,date (full release date), rating
    # (user rating at themoviedatabase.org),
    # poster (image url), overview (movie description))
    with open('gen/movies') as f:
        movies = pickle.load(f)
```

```
    # get reviewer name list
    # These names are fake, though the ratings are biased around the actual rating
    with open('gen/names') as f:
        names = f.read().splitlines()

    # Grab create sql from files. Normally, I would embed it, but this is easier to read.
    with open('gen/drop.sql') as f:
        dropsql = f.read()
    with open('gen/create.sql') as f:
        createsql = f.read()

    try:
        conn.executescript(dropsql)
        print "Tables dropped"
    except sqlite3.OperationalError:
        print "Tables don't exist so not dropping them"

    try:
        conn.executescript(createsql)
        print "Tables created"
    except:
        print "Unable to create tables"

    try:
        # executes the following line for every object in movies
        conn.executemany("insert into movie (title,director,year,poster,overview)\
            values (:title, :director, :year, :poster,:overview);",movies)
        print "Movie data inserted"
    except:
        print "Unable to insert movie data"

    try:
        # executes the following line for every name in list
        conn.executemany("insert into reviewer (name) values (?);",[(x,) for x in names])
        print "Reviewer data inserted"
    except:
        print "Unable to insert reviewer data"

    try:
        # make sure some have no rating
        moviesSkipped = random.sample(xrange(0,len(movies)),random.randint(12,25))

        # for each reviewer... (by index)
        for reviewer in xrange(len(names)):
            # get random sample of movies
            moviesReviewed = random.sample(xrange(0,len(movies)),random.randint(75,125))

            # for each movie... (by index)
            for movie in moviesReviewed:
                if movie in moviesSkipped:
                    continue
                # get randomized rating biased toward real rating (scaled to 5 star system)
                rating = int(round(movies[movie]['rating'] / 2.0 + random.randint(-1,1)))
                # make sure rating is in [0,5]
                if rating < 0:
                    rating = 0
                elif rating > 5:
                    rating = 5
                # get random date within 90 days of release
                ratingDate = movies[movie]['date'] + datetime.timedelta(random.randint(0,90))
                conn.execute("insert into rating (movieID,reviewerID,stars,ratingDate)\
                    values (?,?,?,?);",(movie,reviewer,rating,ratingDate))
        print "Rating data inserted"
    except IndexError:
        print "Unable to insert rating data"

    # push data into database
    conn.commit()

def get_best_worst(conn,order,year=False):
    # get 10 best or worst movies (chosen by order), optionally by year
    if year:
        return conn.execute('select first.movieID as movieID,title,poster,stars,votes from \
            (select movieID,avg(stars) as stars,count(stars) as votes from rating group by movieID order by stars {0}) first \
            join (select movieID,title,poster from movie where year = ?) second \
            on first.movieID = second.movieID limit 10;'.format(order),(year,))
    else:
        return conn.execute('select first.movieID as movieID,title,poster,stars,votes from \
            (select movieID,avg(stars) as stars,count(stars) as votes from rating group by movieID order by stars {0} limit 10) first \
            join (select movieID,title,poster from movie) second \
            on first.movieID = second.movieID;'.format(order))

def get_random(conn):
    # get 10 random movies
    return conn.execute('select first.movieID as movieID,title,poster,stars,votes from \
        (select movieID,title,poster from movie order by random() limit 10) first \
        join (select movieID,avg(stars) as stars,count(stars) as votes from rating group by movieID) second \
```

```
        on first.movieID = second.movieID;')

def get_none(conn,page):
    # returns movie rows that have no rating associated; Grab 11 so we know if there should be another page
    return conn.execute('select movieID,title,poster,0 as stars,0 as votes from movie where movieID not in \
        (select movieID from rating) limit 11 offset ?;',((page-1)*10,))

def get_search(conn,page,search):
    # returns movie rows with title like search; Grab 11 so we know if there should be another page
    return conn.execute('select first.movieID as movieID,title,poster,stars,votes from \
            (select movieID,title,poster from movie where title like ?) first \
        join (select movieID,avg(stars) as stars,count(stars) as votes from rating group by movieID) second \
        on first.movieID = second.movieID limit 11 offset ?;',('%'+search+'%',(page-1)*10))

def get_movie(conn,movieID):
    # returns movie info
    return conn.execute('select * from (select * from movie where movieID = ?) first \
        left join (select movieID as ID,avg(stars) as stars,count(stars) as votes from rating group by movieID) second \
        on first.movieID = second.ID;',(movieID,)).fetchone()

def get_reviews_by_movie(conn,movieID):
    # returns reviews on movie
    return conn.execute('select * from (select reviewerID as ID,stars from rating where movieID = ?) first \
        join reviewer on first.ID = reviewer.reviewerID;',(movieID,))

def get_reviewer(conn,reviewerID):
    # returns reviewer info
    return conn.execute('select * from (select * from reviewer where reviewerID = ?) first \
        left join (select reviewerID as ID,avg(stars) as stars,count(stars) as votes from rating group by reviewerID) second \
        on first.reviewerID = second.ID;',(reviewerID,)).fetchone()

def get_reviews_by_reviewer(conn,reviewerID):
    # returns reviews on movie
    return conn.execute('select movie.movieID as movieID,title,poster,stars from \
            movie join rating on movie.movieID = rating.movieID where reviewerID = ?;',(reviewerID,))

def get_years(conn):
    return [x[0] for x in conn.execute('select year from movie group by year;').fetchall()]
```