

# CS432/532: Final Project Report

**Project Title: Movie-Buzz**

**Team Member(s): Kaur Ramandeep, Mishra Abhimanshu**

## I. PROBLEM

Cinematography industry is an ever-growing industry. It's extremely difficult to find the right movie to watch from the never-ending list of finest movies. We built an application that gives you the list of movies to at a mind-boggling speed with search filter in reviews, ratings and box-office collections.

## II. SOFTWARE DESIGN AND IMPLEMENTATION

We take inspiration from IMDb's web interface and try to fix the limitations it has.

### A. Software Design and NoSQL-Database and Tools Used

Mongo, Flask, flask-mongoengine, Python

MongoDB is a NoSQL database, so each entity can be thought of as a document or collection. There is no relationship between two entities in such a schema. Using database models for movies and users allows us to handle data easily and simplifies how we write out queries. We use flask-mongoengine to connect to mongoDB using Python and Flask. It is built on top of PyMongo. We wrote the queries in Mongo, and then converted them to work with flask-mongoengine. We used Anaconda to create a virtual environment to work in so that we can capture every library and package used into a requirements.txt file which makes it easy for anyone else to run our code.

### B. Supported Queries

For each query type, the first query is the raw query that can be used in a Mogo shell. The second query is generated by converting this raw query to be compatible with flask-mongoengine for our use.

#### Simple Queries

##### I. Create

~Insert a movie  
db.insert\_one(Movie)  
Movie.save()  
~Insert a user  
db.insert\_one(User)  
User.save()

##### II. Select

~Find movie by query in name  
db.Movie.find({'name': {'\$regex': ".\*"+query+".\*"/  
i'}})  
Movie.objects(name\_\_icontains=query)  
~Find movies with particular score  
db.Movie.find({'score': query})

Movie.objects(score=query)

~Find movies with score higher than a threshold  
db.Movie.find({'score': {'\$gte': query}})

Movie.objects(score\_\_gte=query)

~Find movies with box office collection higher than a threshold

db.Movie.find({'box\_office': {'\$gte': query}})

Movie.objects(box\_office\_\_gte=query)

## III. Update

~Add new rating to movie and update overall rating using movie index

db.update\_one({'id': index}, {'\$set': {'all\_scores':  
updated\_scores, 'score': updated\_agg\_score}})

Movie.objects.get(id=index).update(all\_scores=updat  
ed\_scores, score=updated\_agg\_score)

~Add new review to movie using movie index  
db.update\_one({'id': index}, {'\$set': {'reviews':  
updated\_reviews}})

Movie.objects.get(id=index).update(reviews=updated  
\_reviews)

~Update any part of movie information (overwrites  
previous version) using movie index

db.update\_one({'id': index}, {'\$set': {'name':  
updated\_name, 'casts': updated\_casts, 'genres':  
updated\_genres, 'reviews': updated\_reviews,  
'all\_scores': updated\_all\_scores, 'score':  
updated\_score, 'box\_office': updated\_box\_office}})

Movie.objects.get(id=index).update(movie\_data)

## IV. Delete

~Delete a movie using movie index

db.delete\_one({'id': index})

Movie.objects.get(id=index).delete()

~Delete all movies

db.Movie.drop()

Movie.drop\_collection()

~Delete all users

db.User.drop()

User.drop\_collection()

#### Complex Queries

##### I. Select

~Find movie by query in any review

db.Movie.find({'reviews': {'\$regex': ".\*"+query+".\*"/  
i'}})

Movie.objects(reviews=re.compile('.\*'+query+'.\*',  
re.IGNORECASE))

~ Find movie by query in any genre

db.Movie.find({'genres': {'\$regex': ".\*"+query+".\*"/  
i'}})

```
Movie.objects(genres=re.compile('.*'+query+'.*',
re.IGNORECASE))
~ Find movie by query in any cast member name
db.Movie.find({'casts': {'$regex': ".*"+query+".*"}})
Movie.objects(casts=re.compile('.*'+query+'.*',
re.IGNORECASE))
```

These are sophisticated text search queries which are facilitated by Mongo.

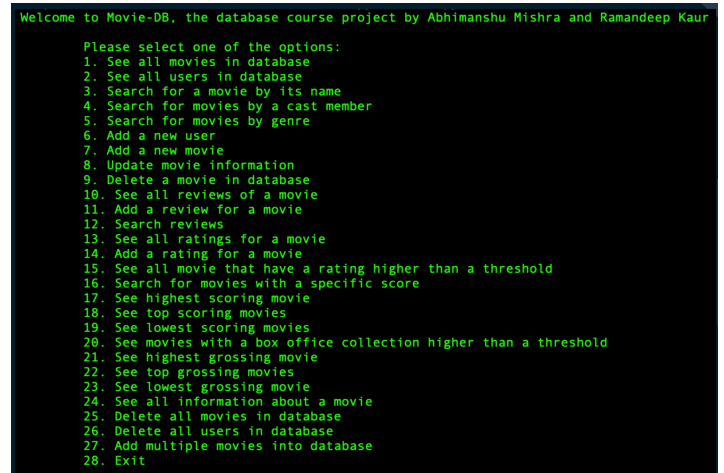
## II. Aggregation

```
~Find any number of movies sorted in ascending
order of box office collection
db.Movie.find().sort({'box_office': 1}).limit(number)
Movie.objects().order_by("box_office").limit(number)
)
~Find any number of movies sorted in ascending
order of rating
db.Movie.find().sort({'score': 1}).limit(number)
Movie.objects().order_by("score").limit(number)
~Find any number of movies sorted in descending
order of box office collection
db.Movie.find().sort({'box_office':
-1}).limit(number)
Movie.objects().order_by("-
box_office").limit(number)
~Find any number of movies sorted in descending
order of rating
db.Movie.find().sort({'score': -1}).limit(number)
Movie.objects().order_by("-score").limit(number)
```

## II. COMMAND LINE INTERFACE

We write the command line interface using Python. A screenshot of the various options our app has to offer is presented below.

A second file contains all the functions that send requests to our locally hosted Flask API. When a user selects an option in the menu and fills in the required information, these utility functions are used to connect to the Flask backend which, in turn, queries the associated MongoDB database and retrieves results.



```
Welcome to Movie-DB, the database course project by Abhimanshu Mishra and Ramandeep Kaur

Please select one of the options:
1. See all movies in database
2. See all users in database
3. Search for a movie by its name
4. Search for movies by a cast member
5. Search for movies by genre
6. Add a new user
7. Add a new movie
8. Update movie information
9. Delete a movie in database
10. See all reviews of a movie
11. Add a review for a movie
12. Search reviews
13. See all ratings for a movie
14. Add a rating for a movie
15. See all movie that have a rating higher than a threshold
16. Search for movies with a specific score
17. See highest scoring movie
18. See top scoring movies
19. See lowest scoring movies
20. See movies with a box office collection higher than a threshold
21. See highest grossing movie
22. See top grossing movies
23. See lowest grossing movie
24. See all information about a movie
25. Delete all movies in database
26. Delete all users in database
27. Add multiple movies into database
28. Exit
```

## III. PROJECT OUTCOMES

- <https://github.com/abhimanshumishra/movie-db>
- <https://www.youtube.com/watch?v=UpAlqQj1XBA>
- <https://docs.google.com/presentation/d/1oI69I6I3GT9I36UzPbxt7Kf0TQG3wSXQHuo9NiW7Nc0/edit?usp=sharing>

## IV. PROJECT SETUP

### I. DATABASE SCHEMA

```
Movie(name, casts, genres, reviews, all_scores, score,
box_office)
User(email, password)
```

## REFERENCES

1. R. Elmasri and S. B. Navathe, Algorithms, Fundamentals of Database Systems, Seventh Edition, Pearson, 2017.
2. Michael Ernst, How to Write a Technical Paper, <https://homes.cs.washington.edu/~mernst/advice/write-technical-paper.html>.
3. <http://docs.mongoengine.org/projects/flask-mongoengine/en/latest/>
4. <https://flask.palletsprojects.com/en/1.1.x/>
5. <https://www.python.org/>