



**Final Project Report**

**Advanced Computer Programming**

**Movie Sentiment Explorer**

**Group : 12312**

**Instructor : DINH-TRUNG VU**

**2025-06-09**

# Introduction

## Group Information

1. **Group Project Repository**:

<https://github.com/ACP-Final-Group12/ACP_Final>

1. **Group members**:
   1. Munkhtselmeg (113021193)
   2. Amarsanaa
   3. Nomin

## Overview

In our final project, *Movie Sentiment Explorer*, we utilized several advanced Python features and libraries to build a complete web-based application that analyzes public sentiment on IMDb movie reviews.

We implemented the following advanced language features and technologies:

* **dataclass**: Used to structure and manage movie metadata as clean, immutable objects.
* **pprint (Pretty Print)**: Helped in debugging and displaying structured outputs during development.
* **pattern matching (match-case)**: Used to simplify logic when handling various sentiment labels.
* **regular expressions (re)**: Applied for cleaning and extracting keywords from raw text data when needed.
* **BeautifulSoup**: Used to scrape user reviews directly from IMDb pages in real time.
* **Flask**: Provided the web framework to handle routes, rendering HTML templates, and serving user requests.
* **Chart.js (JavaScript library)**: Used to dynamically generate visual bar charts comparing IMDb ratings and user sentiment scores.
* **NLTK and** VADER Sentiment Analyzer: Used to analyze sentiment polarity (positive, neutral, or negative) for each review.

Our project successfully takes a movie title as input, scrapes and analyzes its IMDb reviews, computes a sentiment score, and displays both textual and visual summaries. Users can download results as PDF or CSV, view charts, and toggle between dark/light modes. We also implemented a feature to download the movie poster.

# Implementation

## MovieData

The MovieData class is a Python @dataclass used to structure and store metadata about a movie. This class plays a central role in organizing the information retrieved from the OMDb API in a clean and manageable format. By using Python's dataclass feature, it eliminates boilerplate code and automatically provides built-in methods such as \_\_init\_\_, \_\_repr\_\_, and \_\_eq\_\_.

### Fields (Attributes of MovieData)

* **title (str)**: Stores the official title of the movie as returned by the OMDb API. This is displayed prominently in the UI and used as the primary identifier throughout the application.
* **year (str)**: Represents the release year of the movie. It helps distinguish between different versions or remakes of films with similar titles.
* **director (str)**: Contains the name(s) of the director(s) of the movie. This data is useful for users who may want to filter or explore movies based on specific filmmakers.
* **genre (str)**: A comma-separated string describing the movie’s genre(s) such as Action, Drama, Sci-Fi, etc. This is used both for display purposes and potential future features like genre-based filtering or recommendations.
* **plot (str)**: A concise textual summary of the movie’s storyline. This field provides context for the viewer and is displayed directly in the application below the title and poster.
* **imdb\_rating (float)**: A numerical value representing the official IMDb rating for the movie. It is extracted as a float and displayed visually next to the user sentiment rating for easy comparison.
* **poster\_url (str)**: A direct link to the movie's poster image. This URL is used to display the poster in the UI and also allows users to download the image for reference or sharing.

### Methods (Automatically Provided by the MovieData Dataclass)

This dataclass leverages Python’s automatic method generation:

\_\_init\_\_() – Automatically initializes the class with field values.

\_\_repr\_\_() – Provides a string representation of the object.

\_\_eq\_\_() – Supports comparison operations between instances.

### Functions (Structuring Movie Metadata from the OMDb API)

The function fetch\_movie\_data(title: str) -> MovieData utilizes the MovieData class. It fetches data from the OMDb API based on the provided movie title, parses the response, and returns a populated MovieData object, which is then used throughout the system—especially in the web UI to display movie information and to enable download and export features.

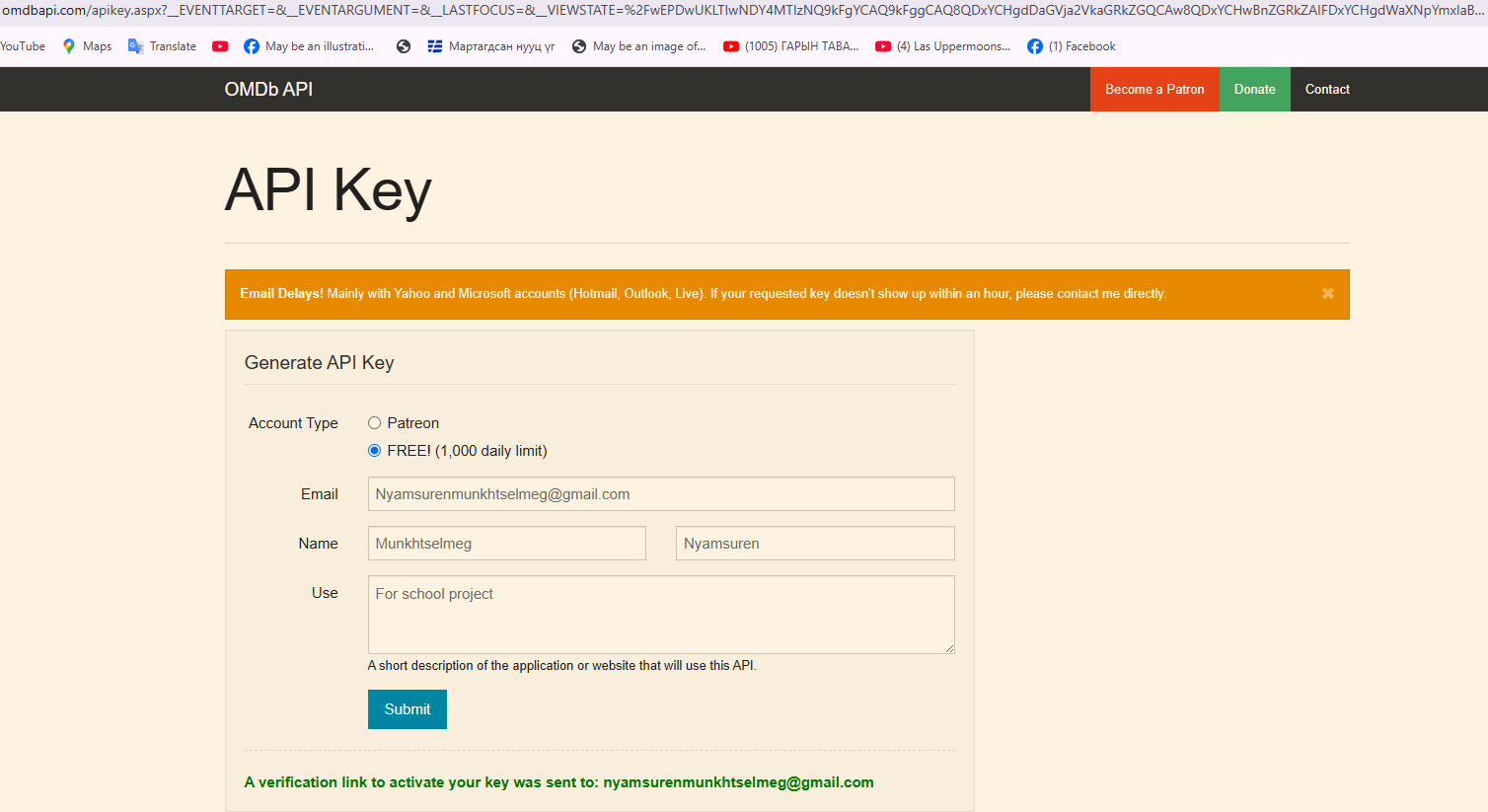
## Class 2 (SentimentAnalyzer)

This module performs **sentiment analysis** on user reviews using the **VADER (Valence Aware Dictionary and sEntiment Reasoner)** model from the nltk.sentiment.vader library. It is written as a set of functions but functions like a class in responsibility and usage.

## Method/Function 1 (Fetching Movie Metadata)

### Purpose

Retrieves structured movie metadata such as title, year, genre, director, plot summary, IMDb rating, and poster URL using the **OMDb API**.



### Key Features & Techniques

* Uses the requests library to call the OMDb API.
* Implements custom User-Agent headers to mimic browser behavior.
* Validates and cleans up API responses before creating a Movie data class instance.

### Returns

A Movie data class object containing all metadata needed for the UI, poster preview, chart rendering, and exporting.

from dataclasses import dataclass

import requests

import os

API\_KEY = os.getenv("586ea014",)

@dataclass

class MovieData:

title: str

year: str

director: str

genre: str

plot: str

imdb\_rating: float

poster\_url: str

def fetch\_movie\_data(title: str) -> MovieData:

url = f"http://www.omdbapi.com/?apikey={API\_KEY}&t={title}"

headers = {"User-Agent": "Mozilla/5.0"}

response = requests.get(url, headers=headers)

data = response.json()

return MovieData(

title=data.get("Title", "N/A"),

year=data.get("Year", "N/A"),

director=data.get("Director", "N/A"),

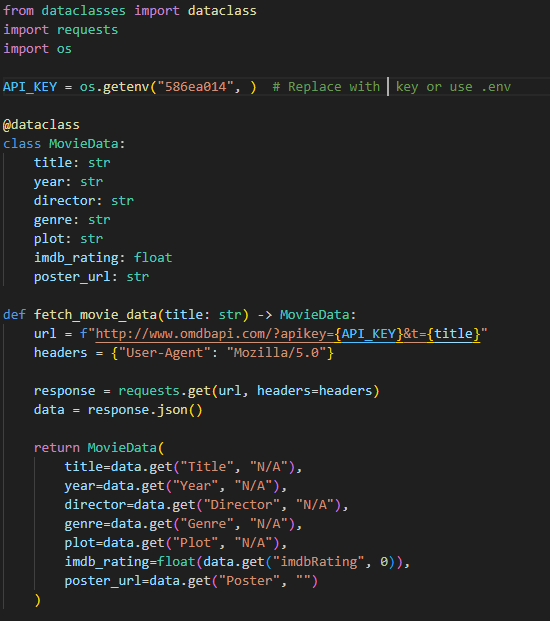
genre=data.get("Genre", "N/A"),

plot=data.get("Plot", "N/A"),

imdb\_rating=float(data.get("imdbRating", 0)),

poster\_url=data.get("Poster", "")

)



## Method/Function 2 ( Web Scraping IMDb Reviews)

### Purpose

Scrapes real user reviews from the IMDb website using **BeautifulSoup** for sentiment analysis.

### Key Features & Techniques

* Uses requests with timeout and headers to fetch HTML pages.
* Parses review blocks using BeautifulSoup with fallback selectors (.text.show-more\_\_control, .ipc-html-content, etc.).
* Filters and cleans text content before processing.

### Returns

A list of raw review strings ready for sentiment scoring and UI display.

import requests

from bs4 import BeautifulSoup

import re

def get\_imdb\_reviews(title: str):

query = title.lower().replace(" ", "+")

search\_url = f"https://www.imdb.com/find?q={query}&s=tt&ttype=ft&ref\_=fn\_ft"

headers = {

"User-Agent": "Mozilla/5.0"

}

search\_resp = requests.get(search\_url, headers=headers, timeout=10)

soup = BeautifulSoup(search\_resp.text, "html.parser")

# Get the first result's IMDb page

first\_result = soup.select\_one("td.result\_text a")

if not first\_result:

return []

movie\_url = "https://www.imdb.com" + first\_result["href"]

reviews\_url = movie\_url.split("?")[0] + "reviews"

# Scrape the reviews page

reviews\_resp = requests.get(reviews\_url, headers=headers, timeout=10)

soup = BeautifulSoup(reviews\_resp.text, "html.parser")

review\_blocks = soup.select(".text.show-more\_\_control, .ipc-html-content")

reviews = []

for block in review\_blocks:

text = block.get\_text(strip=True)

text = re.sub(r"\s+", " ", text)

if text and len(text) > 20:

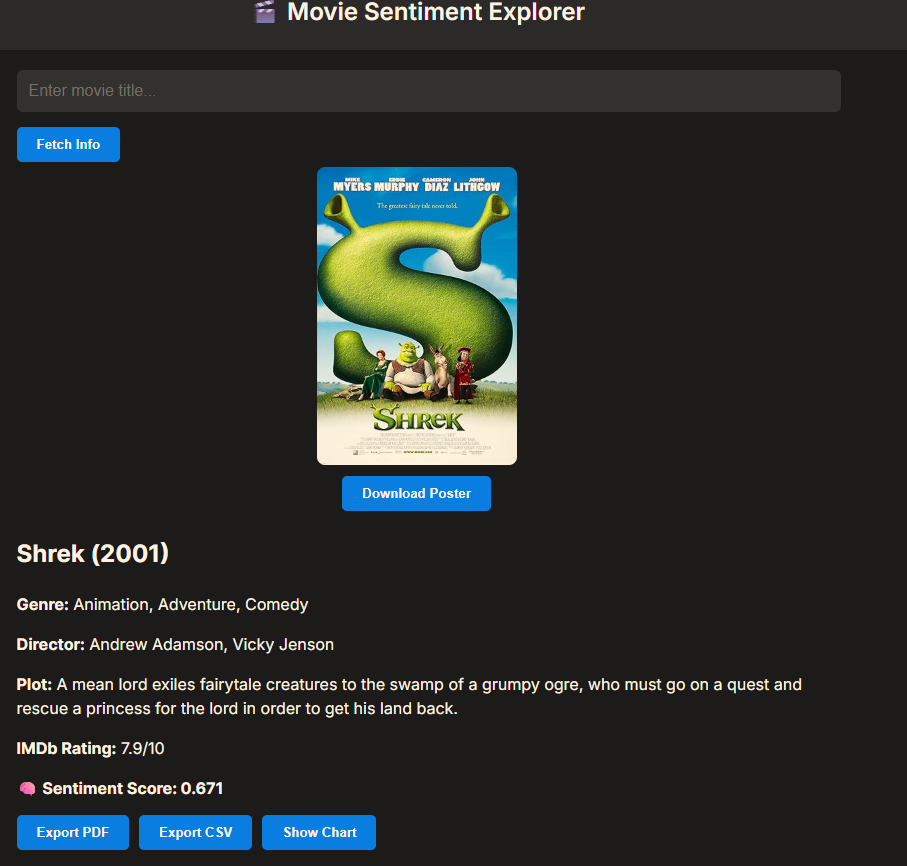
reviews.append(text)

return reviews

# Results

## Result 1 (Movie Information & Poster Preview)

**Description:** After entering a valid movie title like "Inception", the system fetches the movie metadata from the OMDb API. The information includes the title, release year, genre, director, IMDb rating, plot, and poster image.



**Visual Proof:** 🖼️ Include a screenshot showing:

* Movie title
* Poster displayed
* Metadata (year, genre, director, plot)
* IMDb rating

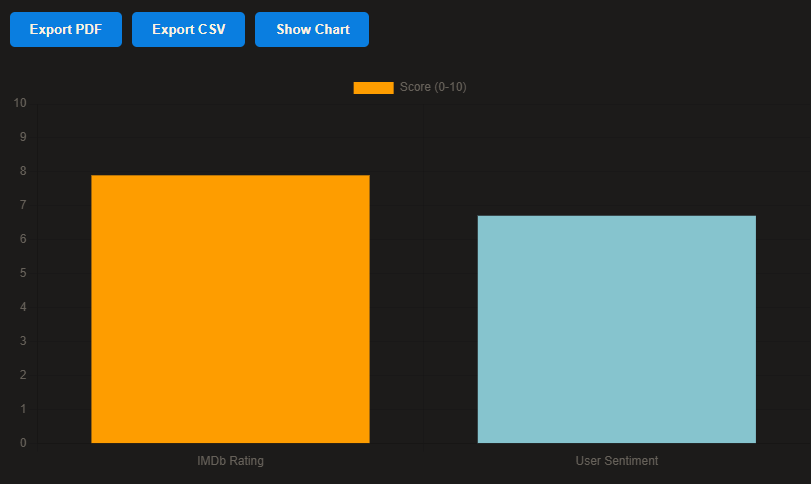
## Result 2 (Sentiment Analysis & Export Functions)

**Description:** The system scrapes real user reviews from IMDb and performs sentiment analysis using VADER. The sentiment score is calculated, categorized, and displayed along with individual reviews. Users can also:

* Toggle dark/light mode
* Show a comparison chart (IMDb rating vs sentiment score)
* Export data as PDF or CSV

**Visual Proof:** 🖼️ Include a screenshot showing:

* Sentiment score
* Reviews with sentiment labels (Positive/Negative/Neutral)
* Chart canvas
* Export buttons



.

# Conclusions

In this project, we successfully developed a full-featured Flask-based web application called **Movie Sentiment Explorer**. The system allows users to input any movie title and receive detailed movie information, IMDb ratings, user reviews, and sentiment analysis results. Additionally, users can export data as PDF and CSV, toggle between light/dark themes, and visualize sentiment through interactive charts.

We utilized several advanced programming concepts such as dataclass structures for organizing movie data, pattern matching for parsing content, regular expression for cleaning text, and beautiful soup for web scraping IMDb reviews. Sentiment analysis was implemented using the vader model from the NLTK library, and data visualization was achieved with [chart.js](http://chart.js) .

Throughout the development, we focused on creating a clean, modern UI and ensuring the system remained responsive and interactive. This project demonstrates our ability to integrate multiple technologies into a single user-friendly application, and it highlights the practical use of web scraping, natural language processing, and Flask-based web design in real-world projects.