



**Final Project Report**

**Advanced Computer Programming**

**Movie Sentiment Explorer**

**Group : 8**

**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 (113021188)
   3. Nomin (113021190)

## Overview

For our final project, we developed **Movie Sentiment Explorer**, a complete web-based application that helps users understand how people feel about movies by analyzing real user reviews from IMDb. Our goal was to go beyond basic star ratings and use AI to uncover the actual sentiment behind public opinions.

As a team, we combined several advanced Python features and libraries to bring this idea to life:

* We used **dataclasses** to structure movie metadata in a clean, organized way.
* **Pattern matching (match-case)** helped us simplify logic when working with different sentiment categories.
* With **regular expressions**, we cleaned and extracted keywords from raw text.
* **BeautifulSoup** allowed us to scrape user reviews directly from IMDb in real time.
* We used **Flask** to build the backend, manage routes, and render HTML templates.
* On the frontend, we integrated **Chart.js** to visualize sentiment data in interactive pie charts.
* For analyzing sentiment, we applied **VADER** from the **NLTK** library to classify reviews as positive, neutral, or negative.

Our application allows users to search any movie title, view its official information via the **OMDb** and **TMDb APIs**, analyze its sentiment score based on real reviews, and explore results visually and interactively. Users can also switch between light and dark themes, export results as PDF or CSV files, and download the movie poster.

This project reflects our teamwork in combining web scraping, sentiment analysis, and full-stack web development into one seamless, user-friendly tool.

# 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.

@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:

response = requests.get(f"http://www.omdbapi.com/?t={title}&apikey=API\_KEY")

data = response.json()

return MovieData(

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

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

...

)

### 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.

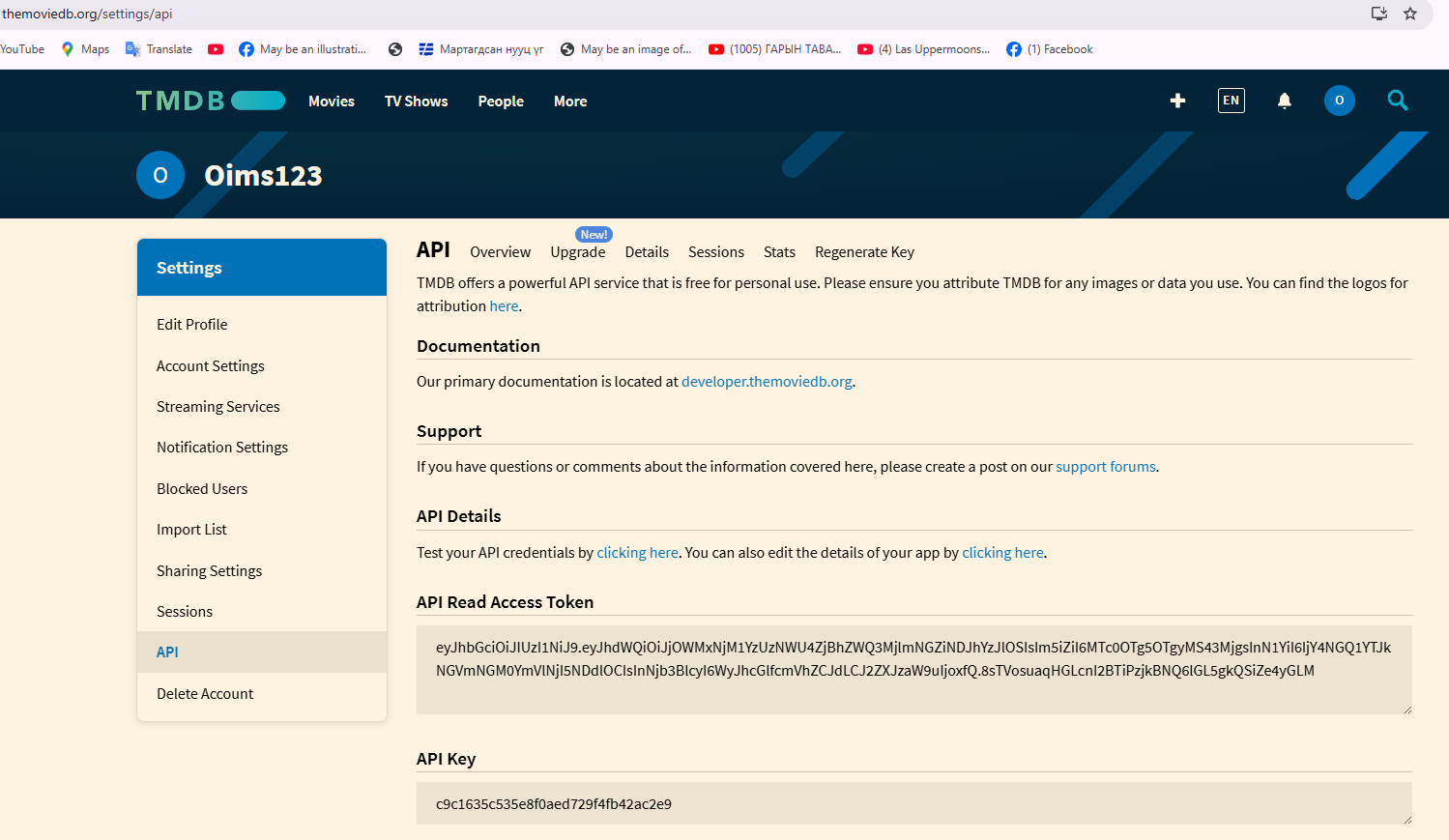
## Class 2 (SentimentAnalyzer)

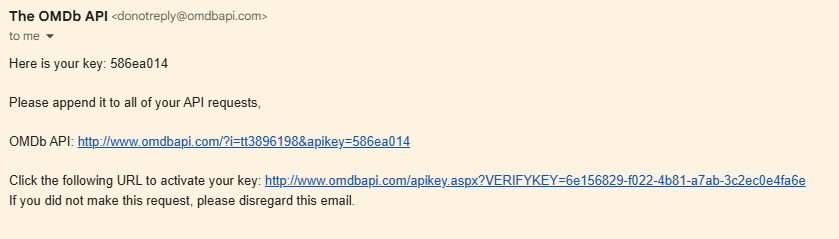
This module performs **sentiment analysis** on user reviews using the **VADER (Valence Aware Dictionary and sEntiment Reasoner)** model from the nltk.sentiment.vader library. It is written as a set of functions but functions like a class in responsibility and usage.

## Method/Function 1 (Fetching Movie Metadata)

### Purpose

This function retrieves structured movie metadata based on a given movie title. It uses the **OMDb API** as the primary data source, and optionally integrates the **TMDb API** to enhance poster quality or provide additional metadata.





### Key Features & Techniques

* Sends an HTTP request to the **OMDb API** using the requests library
* Adds custom headers to simulate browser requests for smoother API interaction
* Parses and validates the JSON response
* When needed, queries the **TMDb API** to retrieve higher-resolution poster images
* Returns a structured MovieData object for frontend use and export features

### Returns

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

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

response = requests.get(f"http://www.omdbapi.com/?t={title}&apikey=API\_KEY")

data = response.json()

# Optional: fallback to TMDb for better poster if needed

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

if not poster or poster == "N/A":

poster = get\_tmdb\_poster(title)

return MovieData(

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

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

...

poster\_url=poster

)

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

### Purpose

This function scrapes real user reviews from the **IMDb website** based on a searched movie title. These reviews serve as the primary input for the sentiment analysis process performed later using VADER.

### Key Features & Techniques

* Uses the requests library to send HTTP requests with custom headers and timeout to prevent blocking
* Parses HTML content using **BeautifulSoup**
* Searches for the movie page by title, then navigates to its review section
* Extracts multiple review blocks using fallback CSS selectors to support varying IMDb layouts
* Cleans and filters the text using **regular expressions**, removing extra whitespace and short/noisy content

### Returns

A list of cleaned review strings. Each review is a full user-written opinion extracted directly from IMDb and ready to be passed into the sentiment analysis pipeline.

These reviews are later:

* Scored using the VADER model
* Shown on the website with their sentiment labels
* Used to calculate the overall sentiment score and feed into the sentiment chart

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

search\_url = f"https://www.imdb.com/find?q={title.replace(' ', '+')}&s=tt"

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

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

...

return cleaned\_reviews

# Results

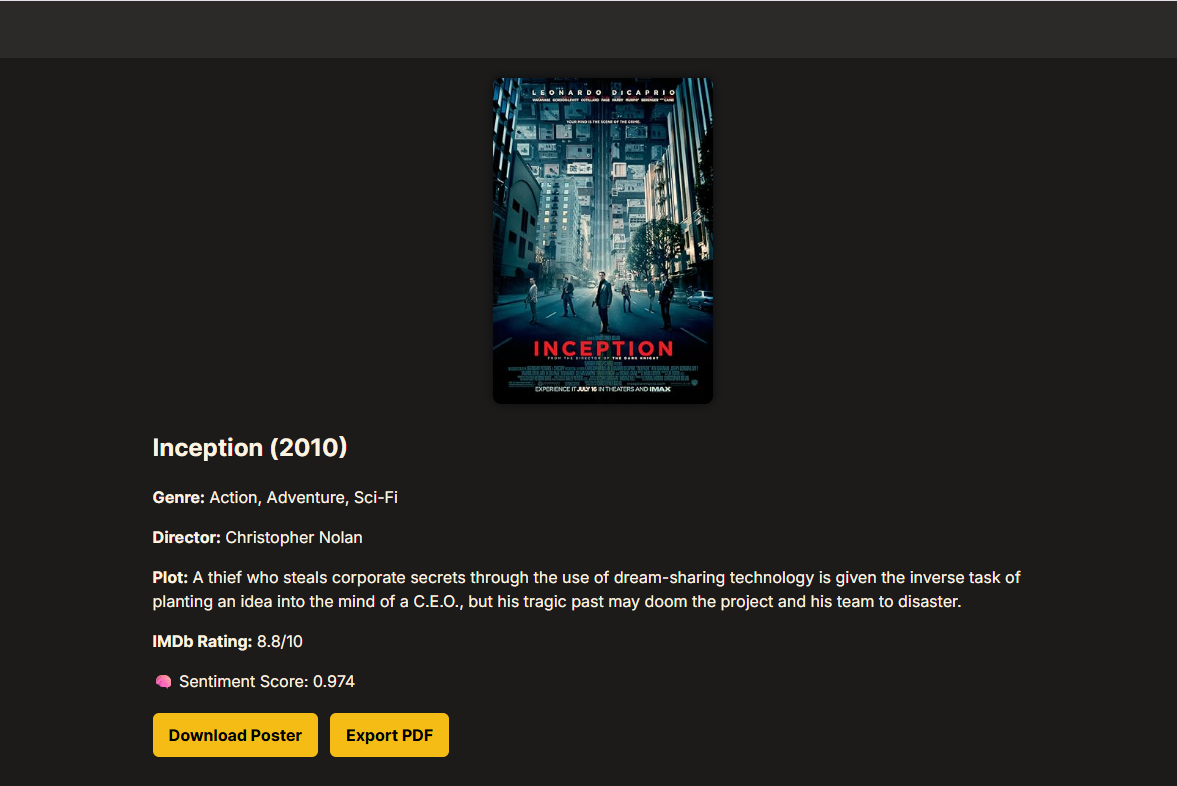
## Result 1 (Movie Information & Poster Preview)

When a user enters a movie title (e.g., *“Inception”*), the system sends a request to the **OMDb API** to retrieve official movie metadata. If needed, it enhances the visual quality of the poster using the **TMDb API**. The retrieved information is then displayed in a clean and user-friendly layout on the website.

The displayed movie metadata includes:

* **Title**
* **Release Year**
* **Genre**
* **Director**
* **Plot Summary**
* **IMDb Rating**
* **Movie Poster**

This section helps users quickly understand the basic context of the movie before diving into its reviews and sentiment analysis.



## Result 2 (Sentiment Analysis & Export Functions)

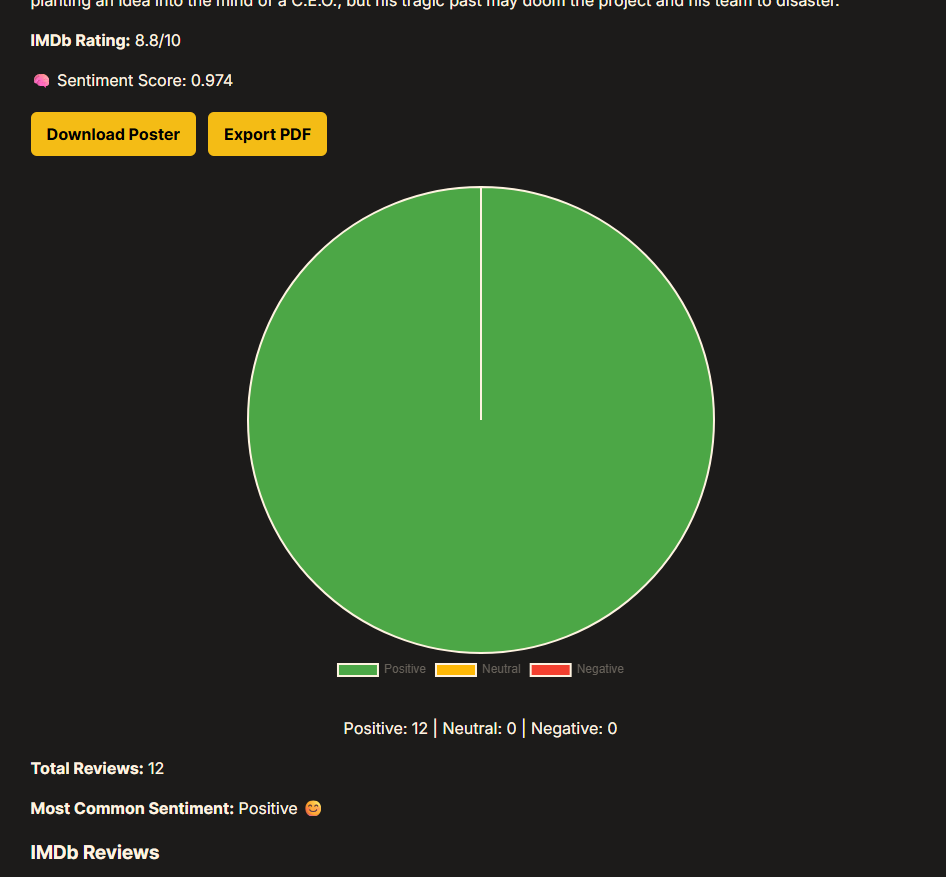
Once a movie is selected, the system automatically **scrapes real user reviews** from IMDb and analyzes them using the **VADER sentiment analyzer** from the NLTK library. Each review is categorized as **positive**, **neutral**, or **negative**, and a **compound sentiment score** is calculated based on the overall distribution.

This result is presented in both **textual** and **visual** formats:

* A numerical **sentiment score** is displayed
* An interactive **Chart.js pie chart** shows sentiment distribution
* Each review is shown with its corresponding sentiment label

Additionally, users have the ability to:

* **Toggle between dark and light themes** for better user experience
* **View a visual comparison** of IMDb rating vs. AI-generated sentiment score
* **Export sentiment results** and movie details as **PDF** or **CSV** files for sharing or offline analysis



.

# Conclusions

In this project, we successfully developed a web-based application called **Movie Sentiment Explorer** that combines web scraping, sentiment analysis, and interactive visualizations to help users evaluate movies based on real user opinions.

Using Python and Flask, we built a responsive and functional system that retrieves official movie data via the **OMDb** and **TMDb APIs**, scrapes user reviews from **IMDb**, and analyzes sentiment using the **VADER** model from NLTK. The results are displayed with a user-friendly interface that includes metadata, reviews, an interactive chart, and export options.

Throughout the project, we applied several advanced programming concepts such as:

* **dataclasses** for structured data management
* **pattern matching** for decision-making
* **regex** for text processing
* **BeautifulSoup** for web scraping
* **Chart.js** for frontend visualization

We also focused on delivering a modern and intuitive user experience by implementing features like light/dark theme toggle and poster downloading.

Overall, this project demonstrates our ability to integrate multiple technologies into a full-stack application that is both technically sound and practically useful. It showcases our skills in Python, web development, data analysis, and teamwork.