UNIVERSIDAD DE GRANADA

Ingeniería de Sistemas de Información

Curso 2023

Documentación final del Proyecto

Sistema Comparador de Películas y Series: Miratrés

Amador Carmona Méndez

Índice:

Propuesta de proyecto

Descripción del ámbito del proyecto

Comparador de calificaciones de diferentes películas y series que han obtenido en distintos sitios web. El objetivo es que el usuario del sistema sea capaz de componer una opinión previa para decidir si ver o no una película/serie o seleccionar una de entre varias opciones. Las funcionalidades concretas serán definidas en el futuro, pero el esquema principal del sistema es el comentado recientemente.

Cabe aclarar que los puntos descritos en el presente documento son, o pueden ser, de carácter provisional, es decir, pueden ser modificados o presentar adiciones/eliminaciones en el futuro (principalmente de cara al establecimiento del coste). Además, es posible que algunos de los puntos de los aspectos comentados a continuación (en cualquiera de las secciones), pasen a pertenecer a otra categoría, a no existir o a ser sustituidos. Ídem para las fuentes de datos.

Estudio Preliminar

Aspectos a tener en cuenta

- Comparar calificaciones de películas y series obtenidas a partir de diferentes fuentes.
- Interfaz visual cómoda visualmente hablando, sencilla e intuitiva.
- Inicio de sesión.
- Buscador de películas y series.
- En cada película y serie habrá un apartado con información general sobre las mismas como sinópsis, reparto, año de estreno, etc.
- Apartado para calificar una película o serie.

Aspectos descartados que no se tendrán en cuenta

- Apartado de comentarios dentro de cada película o serie.
- Visualización de las películas o series.

- Gastos dentro de la aplicación.

Aspectos considerados inicialmente pero propuestos hasta futuras versiones

- Lista para almacenar las películas o series favoritas/vistas del usuario.
- Lista de las películas y series con mayor calificación general o, en su defecto, apartado de recomendaciones.
- Cookies.
- Elementos de mejora de la interfaz visual.

Mercado

Cinematográfico/Plataformas de streaming de video.

Proyectos y sistemas existentes en el mismo ámbito

Flick Metrix - https://flickmetrix.com
IMDB - http://imdb.com
FilmAffinity - https://www.filmaffinity.com

Lista de fuentes de datos que se pretenden utilizar

eCartelera - https://www.ecartelera.com
Sensacine - https://www.sensacine.com
FilmAffinity - https://www.filmaffinity.com
IMDB - http://imdb.com
API de The Movie Database - https://developers.themoviedb.org
API IMDB - https://developer.imdb.com
API YouTube

Arquitectura del sistema

Decisiones del diseño:

El diseño arquitectónico seleccionado es la arquitectura basada en capas. Este modelo nos permite intercambiar implementaciones más fácilmente.. Esto hace que nuestro sistema sea mucho más escalable pudiendo incorporar características de forma sencilla las capas serán las siguientes:

Capa de presentación, en la que implementaremos las diferentes interfaces del proyecto.

Capa de aplicación, en la que llevará a cabo el desarrollo de las utilidades del programa, comparación de rating de películas, etc.

Capa de datos, donde tendremos las herramientas, web scraping, etc,base de datos, etc para obtener los datos de nuestra aplicación.

El diseño de la representación de la arquitectura del sistema es un modelo diseñado por Philippe Kruchten, llamado 4+1, el cual contiene 4+1 vistas: vista lógica, vista de procesos, vista de desarrollo, vista física y escenarios debido a que usaremos google engine app no podremos hacernos cargo de la vista fisica por que no está a nuestro alcance.

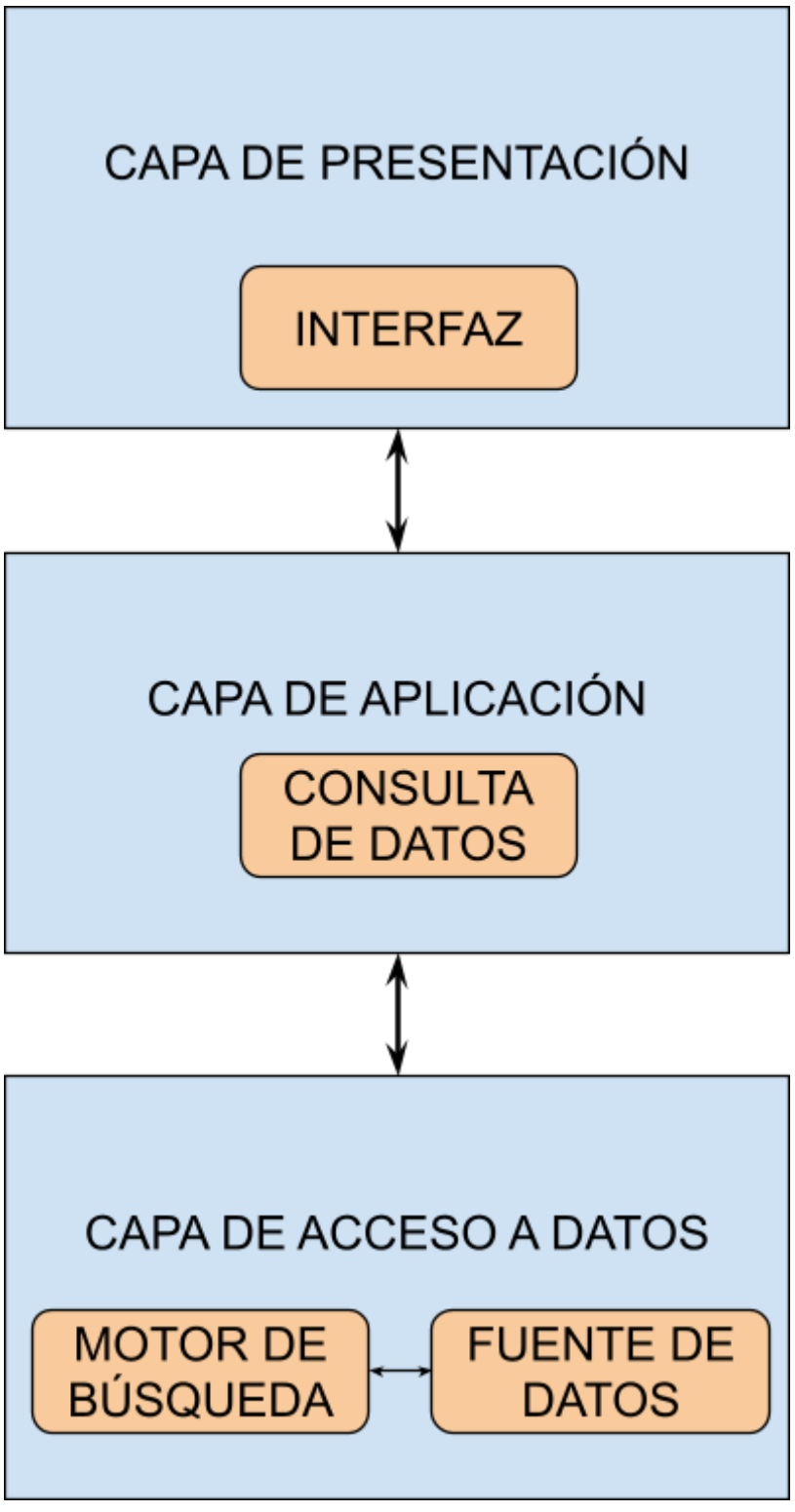
Vista Lógica

La vista lógica describe la estructura del sistema y la funcionalidad de cada una de las partes del sistema. Nuestro sistema está dividido en tres capas, esta sería la estructura del sistema, y la funcionalidad de cada una será:
Capa de presentación, donde estará toda la parte visual de cara al usuario, esta capa será la que comunique con la capa de aplicación y presentará la información de una

forma clara y con un procesamiento mínimo, en nuestro caso seguramente estará implementada con html5 y Bootstrap 5.

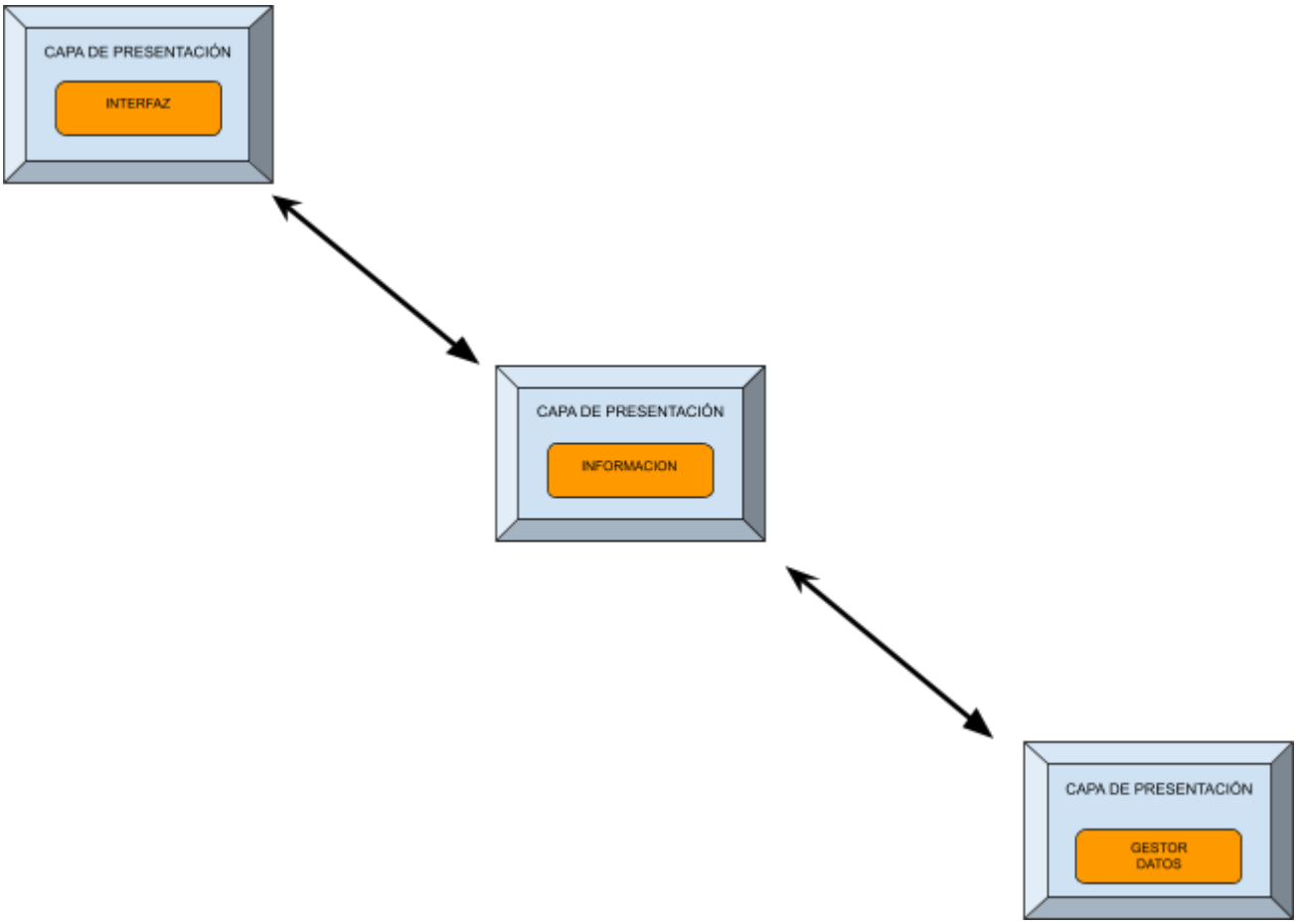
Capa de aplicación, es la capa donde se procesarán todos los datos de nuestra aplicación donde estará el comparador de películas y todas las funcionalidades de nuestra aplicación, en nuestro caso será implementada seguramente con Python y Flask, esta estará comunicada con la capa de datos para obtener los datos y procesarlos y con la capa de presentación para enviar los datos procesados.

Capa de datos, será la capa que obtenga todos los datos de nuestra aplicación aquí utilizaremos web scraping y las APIs, estará conectada con la capa de aplicación para darle los datos a procesar.


CAPA DE PRESENTACIÓN
INTERFAZ
CAPA DE APLICACIÓN
CONSULTA DE DATOS
CAPA DE ACCESO A DATOS
MOTOR DE BÚSQUEDA
FUENTE DE DATOS

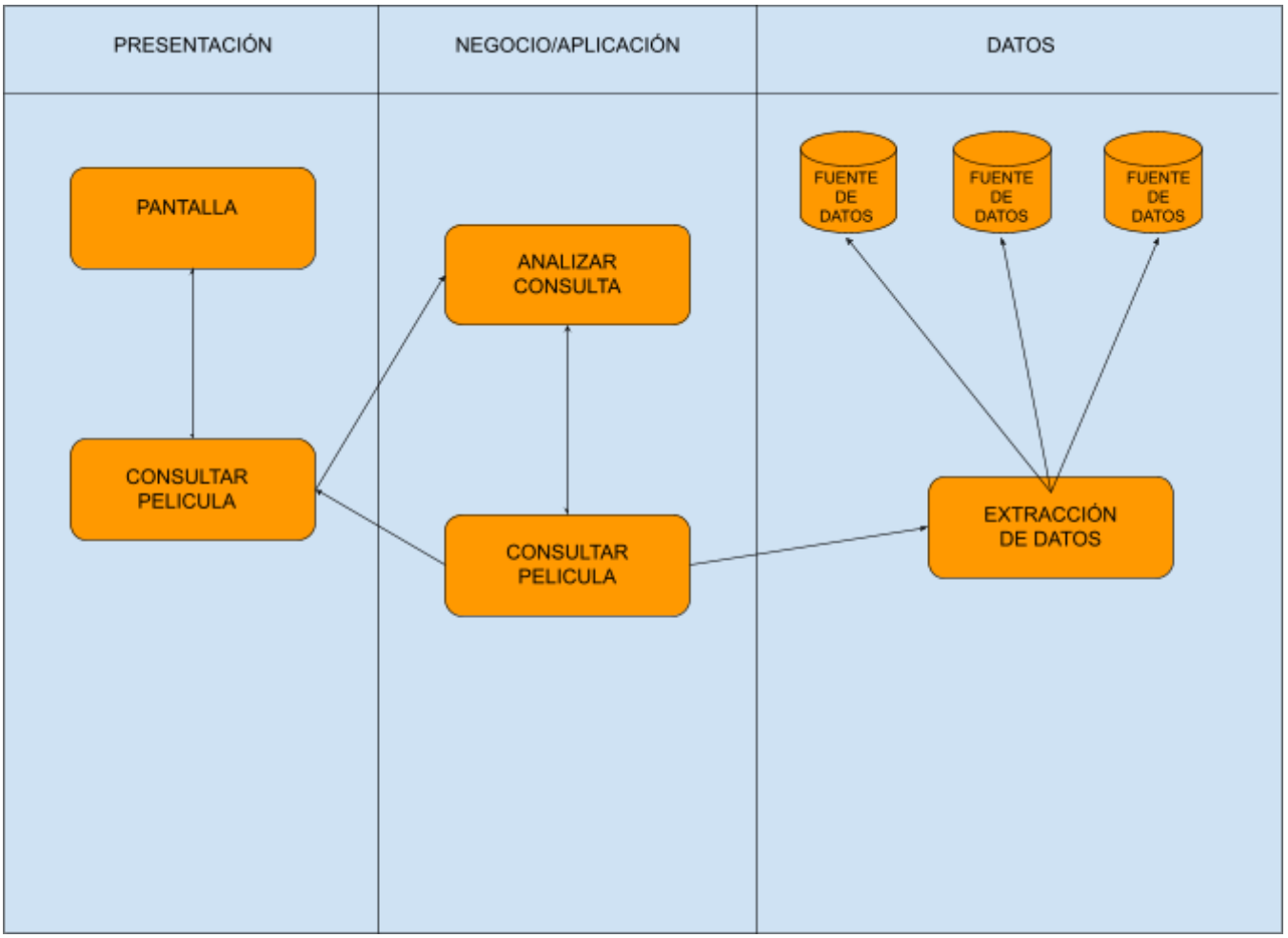
Vista de desarrollo

En la primera capa la capa de presentación estará la parte frontend. En la capa de procesamiento tendremos la parte de backend donde tendremos un módulo para cualquier tipo de consultas y procesamiento de datos .



Vista de Procesos

En esta vista trataremos como los procesos interactúan entre sí, cuales se comunican y cuales deben ocurrir de manera concurrente. en la capa principal habrá un proceso único que será el de extraer y visualizar los datos, es decir proceso frontend, en la capa de aplicación habrá dos procesos que se comunicaran entre sí, uno será para extraer los datos y otro para procesarlos y dárselos a la capa de presentación, y en la capa de datos tendremos tres procesos un proceso de una API, otro de otra api, y otro de web scraping.



Presupuesto del sistema

Presupuesto de desarrollo del sistema

GASTOS DE PERSONAL

Se estima la duración del proyecto en 4 meses dedicados exclusivamente al mismo. El número de responsables en el proyecto será de 3 personas. El total es parcialmente superior al tercio del coste anual, ya que se debe cubrir la correspondencia al período de vacaciones y, además, no todas las horas son facturables.

Contrato temporal de 4 meses 3 Personas sueldo 2200€/mes brutos:

Coste mensual para la empresa por trabajador	**Porcentajes**	**Coste**
Coste total por trabajador		**2951,20€**
Seguridad social a cargo de la empresa	34,6%	761,2€
contingencias comunes	23,60%	519,20€
Prestaciones por desempleo	6,70%	147,40€€
IT/IMS	3,50%	77€
Formación profesional	0,60%	13,20€

FOGASA	0,20%	4,40€

Salario bruto del trabajador	Porcentajes	**2200€**
Impuestos y seguridad social a cargo del trabajador	I.R.P.F. + Cotización a la Seguridad Social	-800,8 €
I.R.P.F.	30%	660,00€
Cotización a la Seguridad Social	6,3%	140,80 €
Contingencias comunes	4,7%	103,40€
Desempleo	1,6%	35,20€
Formación profesional	0,1%	2,20€

Neto: 1399.2€ por trabajador mensualmente.

Total: 35 414,4 €

GASTOS DE EJECUCIÓN

- Costes de adquisición de material inventariable.

Material Inventariable	**Coste (iva incluido)**
Smartphones	3 x 224,6€ = **673,8€**
Portátiles (3 usuarios + 1 test)	4 x 560€ = **2240,00€**
Discos duros	3 x 73,95 = **221,85€**
Monitores (2ª pantalla)	3 x 120 = **360,00€**
TOTAL	**2935,65€**

- Costes de adquisición de material fungible.

Material fungible	**Coste (iva incluido)**
Folios (x5 paquetes de 500)	**21,34€**
Bolígrafos (x50 vic)	**12,99€**
Grapadora (1 unidad)	**3,80€**
Archivador (1 unidad)	**2,14€**
TOTAL	**40,27€**

- Costes de contratos, patentes, licencias, consultoría, suministros…

Material fungible	**Coste (iva incluido)**
Licencias Windows	**259 x 3 = 777€**

Registrar el dominio web con IONOS	**24€/año**
TOTAL	**801€**

GASTOS COMPLEMENTARIOS

- Gastos de desplazamiento, viajes, estancias y dietas.

Se estima una cantidad estimada de cinco reuniones, durante los cuatro meses de desempeño del proyecto, que impliquen desplazamientos. Esto generaría la necesidad de una cantidad de 4500€ para cubrir los gastos de desplazamiento, estancia y dietas de los tres trabajadores de todos los viajes que se necesiten realizar.

- Contratos y alquileres.

Suponiendo que el desarrollo de este producto va a durar 4 meses y estimando que el alquiler de un estudio en Granada cuesta alrededor de 300 euros al mes, estimamos una cantidad de 1200€ destinados para el alquiler de la oficina.

También habría que sumar los gastos de luz y agua de la oficina. Haciendo unos cálculos estimados, el dinero destinado a la luz sería una cantidad de unos 50-60€ al mes, y de agua serían unos 15-20€.

- Gastos de material de difusión, publicaciones, promoción, catálogos, folletos, cartelería…

En publicidad utilizaremos google ads. Estimaremos una cantidad de 10€ al día, lo cual equivale a una cantidad de 1200€ por los cuatro meses.

Presupuesto de despliegue del sistema:

Suponemos que tendremos un tráfico de 1800 usuarios al día, lo que supone al mes unos 54000 usuarios.

InHome	Precio	Google App Engine	Precio
HPE ProLiant DL20 Gen10 Servidor Intel Xeon E-2224/8GB	**1033.99 x 2 = 2067.98€**	**Máquinas virtuales:** 2 VM class: 1,460 total de horas al mes VM class: regular Tipo de instancia: e2-standard-2 USD 97.84 Sistema Operativo / Software: gratuito	**USD 97.84x2 al mes**
Seagate BarraCuda 3,5" 2TB SATA 3	**44,85 X 2 = 89,70€**	**Almacenamiento:** Almacenamiento total: 2,048 TiB Operaciones de clase A*: 45 millones Operaciones de clase B*: 45 millones	**USD 283.96 al mes**
TendaF3 Router Inalámbrico 3000Mbps	**13,00€**	**Cloud SQL:** 730.0 total de horas al mes Almacenamiento SSD: 2,048.0 GiB Backup: 2,048.0 GiB	**USD 561.31 al mes**
Router con balanceador de carga: TP_LINK TL-ER6020	**128,00€**		

TOTAL	**2298,68€+300€ al mes***	**TOTAL**	**1040.95$/mes 940,97 €/mes**

Operaciones de clase A*: Operaciones de lectura y Consulta
Operaciones de clase B*: Operaciones de escritura

*Al total del presupuesto in home habría que sumarle la luz que consume el servidor y los equipos, la conexión a internet por parte de un ISP y la IP fija que estimamos en 300€ al mes.

Coste inicial por usuario:

-in home: 0,0425681481€

-cloud: 0€

Coste por mes por usuario:

-in home: 0.00555555555€

-cloud: 0.01742537037€

Creemos que cloud es la opción más económica.

Total del presupuesto: 50095,2 €

Para el desarrollo de nuestro proyecto de prácticas utilizaremos Amazon Web Service por lo que nos ofrece gratuitamente y ya que personalmente creo que es más fácil de desplegar una aplicación en esta plataforma.

Desarrollo de la aplicación y documentación:

Funcionalidades

La aplicación web consiste en un buscador de películas que nos da el rating en filmaffinity y en themoviedb, así como el trailer de dicha película en youtube.

Cambios respecto a la ordinaria

En la ordinaria había planteado una base de datos o algún tipo de almacenamiento caché para guardar e integrar los datos, pero para desarrollar el proyecto finalmente he decidido que los resultados de la consulta de datos irían directamente, pasando por el main, a la interfaz sin ser almacenados.

Además he añadido como fuente de datos la API de youtube para ya que la mayoría de páginas de rating de películas tienen configuraciones en las que es muy difícil hacer el web scraping por las sesiones de los navegadores y las estructuras que tiene.

Código

El desarrollo de la aplicación web se ha realizado con python, html bootstrap 5 y flask y JavaScript. Para depurar y le he depurado en localhost, a través de WSGI. Después se ha subido el proyecto a Google Platform.

Procesamiento de datos:

La parte de python de la obtención de datos y su procesamiento está implementada en python en el archivo main.py donde esta el procesamiento y la llamadas a los módulo donde se extraen los datos

Web Scraping

El siguiente código utiliza web scraping para obtener información sobre una película específica desde la página de Filmaffinity. Se construye una URL de búsqueda con el nombre de la película y se realiza una solicitud GET para obtener el contenido HTML de la página de búsqueda. Luego, utilizando BeautifulSoup, se analiza el HTML y se busca el primer

resultado de la búsqueda de la película. Si se encuentra el resultado, se extrae el enlace, la imagen, el rating y el título de la película. Finalmente, se devuelve esta información en una tupla. Si no se encuentra ningún resultado, se muestra un mensaje indicando que no se encontraron resultados. El ejemplo de uso al final del código muestra cómo utilizar esta función para buscar información sobre una película ingresada por el usuario y muestra los detalles si se encuentra una coincidencia.

```python
import requests
from bs4 import BeautifulSoup

def buscar_pelicula_filmaffinity(nombre_pelicula):
    # Construir la URL de búsqueda
    nombre_pelicula=nombre_pelicula.replace(" ", "+")
    url_busqueda =
f'https://www.filmaffinity.com/es/search.php?stext={nombre_pelicula}'

    # Realizar la solicitud GET a la página de búsqueda
    response = requests.get(url_busqueda)
    print(url_busqueda)
    # Comprobar si la solicitud fue exitosa
    if response.status_code == 200:
        # Obtener el contenido HTML de la página de búsqueda
        html = response.text

        # Crear un objeto BeautifulSoup para analizar el HTML
        soup = BeautifulSoup(html, 'html.parser')

        # Buscar el primer resultado de la búsqueda
        resultado = soup.find('div', class_='movie-card')

        if resultado:
            # Extraer el enlace, la imagen y el rating de la película
            enlace = resultado.find('a')['href']
            imagen = resultado.find('img')['src']
            rating = resultado.find('div', class_='avgrat-box').text.strip()
            titulo = resultado.find('a')['title']

            return enlace, imagen, rating, titulo
        else:
            print('No se encontraron resultados para la película.')
    else:
        print('No se pudo acceder a la página de búsqueda.')

# Ejemplo de uso
```

```
"""nombre_pelicula = input('Ingrese el nombre de la película: ')
resultado = buscar_pelicula_filmaffinity(nombre_pelicula)
if resultado:
    enlace, imagen, rating, titulo = resultado
    print('Enlace:', enlace)
    print('Imagen:', imagen)
    print('Rating:', rating)
    print('Título:', titulo)

"""
```

API

He utilizado dos APIs: En la primera realizamos web scraping en la página de Filmaffinity para buscar información sobre una película. Construimos una URL de búsqueda con el nombre de la película, realiza una solicitud GET para obtener el contenido HTML de la página de búsqueda y utiliza BeautifulSoup para analizar el HTML y buscar el primer resultado de la búsqueda de la película. Si se encuentra el resultado, extrae el enlace, la imagen, el rating y el título de la película, y devuelve esta información en una tupla.

Python

```python
import requests

def obtener_informacion_pelicula(nombre_pelicula, api_key):
    # Construir la URL de la API con el nombre de la película y la clave de API
    url = f"https://api.themoviedb.org/3/search/movie?api_key={api_key}&query={nombre_pelicula}"

    # Realizar la solicitud GET a la API
    response = requests.get(url)

    # Verificar si la solicitud fue exitosa
    if response.status_code == 200:
        datos_pelicula = response.json()

        # Verificar si se encontró la película
        if datos_pelicula["total_results"] > 0:
```

```
            # Obtener el primer resultado de la lista de películas
            pelicula = datos_pelicula["results"][0]
            titulo = pelicula["title"]
            rating = pelicula["vote_average"]
            imagen = "https://image.tmdb.org/t/p/w500" +
pelicula["poster_path"]

            return titulo, rating, imagen
        else:
            return None
    else:
        print("Error al realizar la solicitud a la API de The Movie
Database.")
        return None

# Nombre de la película a buscar
"""nombre_pelicula = input("Ingresa el nombre de la película: ")

# Clave de API de The Movie Database
api_key = "0cf8bcf44c54573b1364140cd12ac30f"

# Obtener la información de la película
informacion_pelicula = obtener_informacion_pelicula(nombre_pelicula,
api_key)

if informacion_pelicula:
    titulo, rating, imagen = informacion_pelicula
    print("Título:", titulo)
    print("Rating:", rating)
    print("Imagen:", imagen)
else:
    print("No se encontró información para la película especificada.")
"""
```

El segundo código utiliza la API de YouTube para obtener el enlace del trailer de una película ingresada. Crea una instancia de la API de YouTube con una clave de desarrollador, realiza una búsqueda en YouTube con el nombre de la película y el término "trailer", y obtiene el ID del video del primer resultado de la búsqueda. Luego, construye el enlace del trailer utilizando el ID del video y devuelve este enlace. Si no se encuentra ningún trailer, muestra un mensaje indicando que no se encontró ningún trailer para esa película.

```python
from googleapiclient.discovery import build

def obtener_trailer(nombre_pelicula):
    # Crea una instancia de la API de YouTube
    youtube_api = build('youtube', 'v3',
developerKey='AIzaSyCnhYcwC1FaQmxQVLQr2J48ufkCsbV4P68')

    # Realiza una búsqueda en YouTube con el nombre de la película y el
término "trailer"
    resultado_busqueda = youtube_api.search().list(q=f'{nombre_pelicula}
trailer', part='id', maxResults=1).execute()

    # Obtiene el ID del video del primer resultado de la búsqueda
    if resultado_busqueda['items']:
        video_id = resultado_busqueda['items'][0]['id']['videoId']
        return f'https://www.youtube.com/watch?v={video_id}'
    else:
        return 'No se encontró ningún trailer para esa película'
"""
# Ejemplo de uso
nombre_pelicula = input('Ingresa el nombre de la película: ')
trailer_url = obtener_trailer(nombre_pelicula)
print(f'Trailer de "{nombre_pelicula}": {trailer_url}')
"""
```

Integración de API y Web-Scraping:

Para integrar los resultados de las fuentes de datos en nuestra aplicación y mostrarlos utilizamos Flask. Al acceder a la página de inicio, se muestra un formulario para ingresar el término de búsqueda. Cuando se envía el formulario, se ejecutan tres funciones: una realiza web scraping en Filmaffinity para obtener detalles de la película, otra utiliza la API de themoviedb para obtener información adicional y la tercera utiliza la API de YouTube para obtener el enlace del trailer. Los resultados se muestran en una plantilla "resultado.html". Si no se ingresa un término de búsqueda o no se encuentra la película, se muestra una plantilla "error.html". Si el usuario hace clic en "Volver", se redirige a la página de inicio.

```python
import modules.webscrapingFilmmaffinity as Filmaffinity
import modules.APIthemoviedb as APImovie
import modules.APIyoutube as APIyoutube
```

```
from flask import Flask, render_template, request, redirect

app = Flask(__name__)

@app.errorhandler(500)
def server_error(error):
    return render_template('error.html')

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/buscar', methods=['POST'])
def buscar():
    # Obtener los datos del formulario
    termino_busqueda = request.form.get('termino_busqueda')
    api_key = "0cf8bcf44c54573b1364140cd12ac30f"
    # Ejecutar el script de Python y obtener los resultados
    if(termino_busqueda == "" ):
        return render_template('error.html')
    else:
        item1 = Filmaffinity.buscar_pelicula_filmaffinity(termino_busqueda)

        item2 = APImovie.obtener_informacion_pelicula(termino_busqueda,
api_key)

        item3= APIyoutube.obtener_trailer(termino_busqueda)
        #Pasar los resultados a la plantilla resultado.html
        resultado=[item1,item2,item3]
        return render_template('resultado.html',resultado=resultado)

@app.route("/volver")
def volver():
    return render_template('index.html')


if __name__ == '__main__':
    app.run(debug=True)
```

Apariencia

La apariencia de la aplicación la hemos conseguido con bootstrap que es un framework que nos permite de una forma sencilla darle aspecto a nuestra pagina web, también hemos

utilizado css pero en menor medida, luego hemos usado flask para obtener los datos y mostrarlos y js para un cambio de apariencia de modo dia/noche.
Muestro el codigo del resultado.html ya que aquí es donde podemos ver como utilizamos flask, js, bootstrap y css, los demás html son similares.

```
Unset
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="UTF-8">
            <title>Mira3</title>
            <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0/dist/css/bootstrap.min.cs
s">
                <style>
                    body {
                        background-color: #e6f2ff;
                    }

                .title {
                    color: #ff6666;
                }

                .search {
                    background-color: #ccffff;
                }

                .btn-night {
                    background-color: #1a1a1a;
                    color: #8fbc8f;
                    border-color: #8fbc8f;
                }

                .btn-night:hover {
                    background-color: #8fbc8f;
                    color: #1a1a1a;
                    border-color: #1a1a1a;
                }

                .night-mode {
                    background-color: #2b2b2b;
                    color: #8fbc8f;
                }

                .night-mode .title {
                    color: #8fbc8f;
                }
```

```
                .night-mode .search {
                    background-color: #4d4d4d;
                }
                .volver-message {
                    font-size: 24px;
                    color: #ff6666;
                    margin-top: 30px;
                }
                .movie-image {
                    width: 200px; /* Ancho deseado */
                    height: 300px; /* Alto deseado */
                    object-fit: cover; /* Ajustar la imagen dentro del
contenedor */
                }
                </style>
    </head>
    <body>
        <div class="container-fluid">
            <div class="row mt-5">
                <div class="col-12 text-center">
                    <h1 class="title">MIRA3</h1>
                </div>
                <div class="col-12 mt-5">
                    <div class="row">
                        <div class="col-md-6">
                            <h2>Filmaffinity</h2>
                            <div class="col-12 mt-5">
                                <div class="row">
                                    <div class="col-md-4">
                                        <h4>Cartel</h4>
                                        <img src="{{ resultado[0][1] }}"
alt="Imagen de la película" class="movie-image">
                                    </div>
                                <div class="row">
                                    <div class="col-md-4">
                                        <h4>Título</h4>
                                        <p>{{ resultado[0][3] }}</p>
                                    </div>
                                <div class="row">
                                    <div class="col-md-4">
                                        <h4>Rating</h4>
                                        <p>{{ resultado[0][2] }}</p>
                                    </div>
                                </div>
                            </div>
```

```
                        </div>
                        <div class="col-md-6">
                            <h2>The Movie database</h2>
                            <div class="col-12 mt-5">
                                <div class="row">
                                    <div class="col-md-4">
                                        <h4>Cartel</h4>
                                        <img src="{{ resultado[1][2] }}"
alt="Imagen de la película" class="movie-image">

                                    </div>
                                </div>
                                <div class="row">
                                    <div class="col-md-4">
                                        <h4>Título</h4>
                                        <p>{{ resultado[1][0] }}</p>
                                    </div>
                                </div>
                                <div class="row">
                                    <div class="col-md-4">
                                        <h4>Rating</h4>
                                        <p>{{ resultado[1][1] }}</p>
                                    </div>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>

                <div class="col-11 mt-5 text-end">
                    <button class="btn btn-night btn-sm position-absolute
top-0" id="night-mode-btn">Modo Noche</button>
                </div>
            </div>
        </div>
        <div class="col-12 text-center">
            <h1>Trailer de YouTube</h1>
            <div class="embed-responsive embed-responsive-16by9">
                <iframe class="embed-responsive-item" src="{{ resultado[2]
}}" allowfullscreen></iframe>

            </div>
        </div>
        <div class="col-12 text-center">

            <a href="{{ resultado[2] }}" > "{{ resultado[2] }}"</a>

        </div>
```

```
        <div class="col-12 text-center">
            <div class="error-message">Volver a buscar</div>
            <form action="/volver" method="GET">
                <button type="submit" class="btn btn-primary
mt-3">Volver</button>
            </form>
        </div>
        <script>
            function applyNightMode() {
                const body = document.querySelector("body");
                const title = document.querySelector(".title");
                const search = document.querySelector(".search");
                const nightModeBtn =
document.querySelector("#night-mode-btn");

                const nightMode = localStorage.getItem("nightMode");
                if (nightMode === "on") {
                    body.classList.add("night-mode");
                    title.classList.add("night-mode");
                    search.classList.add("night-mode");
                    nightModeBtn.textContent = "Modo Día";
                } else {
                    body.classList.remove("night-mode");
                    title.classList.remove("night-mode");
                    search.classList.remove("night-mode");
                    nightModeBtn.textContent = "Modo Noche";
                }
            }

        function toggleNightMode() {
            const nightMode = localStorage.getItem("nightMode");
            if (nightMode === "on") {
                localStorage.setItem("nightMode", "off");
            } else {
                localStorage.setItem("nightMode", "on");
            }
            applyNightMode();
        }

        const nightModeBtn = document.querySelector("#night-mode-btn");
        nightModeBtn.addEventListener("click", toggleNightMode);

        // Aplicar el estilo del modo nocturno al cargar la página
        document.addEventListener("DOMContentLoaded", applyNightMode);
            </script>

    </body>
</html>
```

URL de la aplicación:

https://constant-cubist-391912.ew.r.appspot.com