Laboratorio 1
Desarrollo de una API

Redes y Sistemas Distribuidos

Integrantes: Ignacio Gomez,
Mauricio Beckford
y Wilfredo Avila

Objetivos:

🎯 Uso y manejo de API’s.

🎯 Fluidez con Python

(y buenas prácticas también)

El proyecto está dividido en *5 partes*:

1. *Configuración del Entorno*
2. *Construcción de la API Principal*
3. *Integración con API de Feriados*
4. *Evaluación de la API.*
5. *Conclusiones*

1.
Configuración del Entorno e Instalación de librerías

¿Qué es un Entorno Virtual?

(módulo *venv* Python)

Según la documentación oficial, Un entorno cooperativamente aislado de ejecución que permite a los usuarios de Python y a las aplicaciones instalar y actualizar paquetes de distribución de Python sin interferir con el comportamiento de otras aplicaciones de Python en el mismo sistema.

Utilizar un entorno virtual de programación es importante por varias razones clave: Insolación de dependencias, Reproductibilidad de entorno, Facilita actualización de dependencias, Gestiona versionado de Python, Ahorro de espacio y limpieza, etc.

How to install and use
El Entorno Virtual en Python


```
#! Iniciar entorno virtual
$ python3 -m venv .venv #

#! Activarlo y usarlo
$ source .venv/bin/activate

#! Desactivarlo
$ (.venv) source .venv/bin/activate
```

How to use
El Sistema de gestión de paquetes de python


```
#! Activar el virtual env
$ source .venv/bin/activate

#! Activar el virtual env
$ (.venv) pip install -r
  requirements.txt

#! Controlar estilos en desarrollo
$ (.venv) pip install pep8
```

¿Por qué una API?

- 👍 Interoperabilidad/Independencia
- 👍 Escalabilidad
- 👍 Flexibilidad
- 👍 Seguridad
- 👍 Documentación

```
blinker==1.7.0
certifi==2024.2.2
charset-normalizer==3.3.2
click==8.1.7
exceptiongroup==1.2.0
Flask==3.0.2
idna==3.6
iniconfig==2.0.0
```

requirements.txt

Guia rápida de Python con Flask

Aplicación Básica

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"

```

Correr en desarrollo

```
$ export FLASK_APP=main.py
$ export FLASK_ENV=development
$ flask run
```

app.route()

Create –> POST
Read —-> GET
Update -> PUT
Delete —>DELETE

‘/peliculas/<int:id>’

Decorador Ruta+Reglas Método

```
13  @app.route('/peliculas', methods=['GET'])
14  def obtener_peliculas():
15      return jsonify(peliculas)
```

Function Hook

devolución en
formato JSON

app.route()

```python
@app.route('/peliculas', methods=['GET'])
def obtener_peliculas():
    return jsonify(peliculas)

@app.route('/peliculas', methods=['POST'])
def agregar_pelicula():
    # Extract new data from user...
    return jsonify(nueva_pelicula), 201

@app.route('/peliculas/<int:id>', methods=['PUT'])
def actualizar_pelicula(id):
   # Extract new upgraded from user...
   return jsonify(pelicula), 200
   # But if we have an error for upgrade...
   return jsonify({'mensaje': 'Película no encontrada'}), 404

@app.route('/peliculas/<int:id>', methods=['DELETE'])
def eliminar_pelicula(id):
   # Seach the film...
   return jsonify({'mensaje': 'Película eliminada'}), 200
```

El formato **JSON**

Java**S**cript **O**bject **N**otation es un formato basado en texto estándar para representar datos estructurados en la sintaxis de objetos de JavaScript utilizado para transmitir datos en aplicaciones web.

2.
Construcción de la API Principal

Se implementan las siguientes funcionalidades a la API

- ⚙️ **Buscar** la película por su ID y **devolver** sus detalles.
- ⚙️ **Buscar** la película por su ID y **actualizar** sus detalles.
- ⚙️ **Buscar** la película por su ID y **eliminarla**.
- ⚙️ **Devolver** el listado de películas de un **género** específico.
- ⚙️ **Devolver** la lista de películas que tengan determinado string en el **título**.
- ⚙️ **Sugerencia** de una película **aleatoria**.
- ⚙️ **Sugerencia** de una película **aleatoria según género**.

Nota: Para la implementación de los 2 últimos puntos se recurrió a la librería *random*.

En *main.py* ahora tenemos...

```python
@app.route('/peliculas', methods=['GET'])
def obtener_peliculas():
    return jsonify(peliculas)


@app.route('/peliculas/<int:id>', methods=['GET'])
def obtener_pelicula(id):
    pelicula = next((p for p in peliculas if p['id'] == id), None)
    if pelicula:
        return jsonify(pelicula), 200
    return jsonify({'mensaje': 'Película no encontrada'}), 404


@app.route('/peliculas', methods=['POST'])
def agregar_pelicula():
    nueva_pelicula = {
        'id': obtener_nuevo_id(),
        'titulo': request.json['titulo'],
        'genero': request.json['genero']
    }
    peliculas.append(nueva_pelicula)
    print(peliculas)
    return jsonify(nueva_pelicula), 201
```

En *main.py* ahora tenemos...

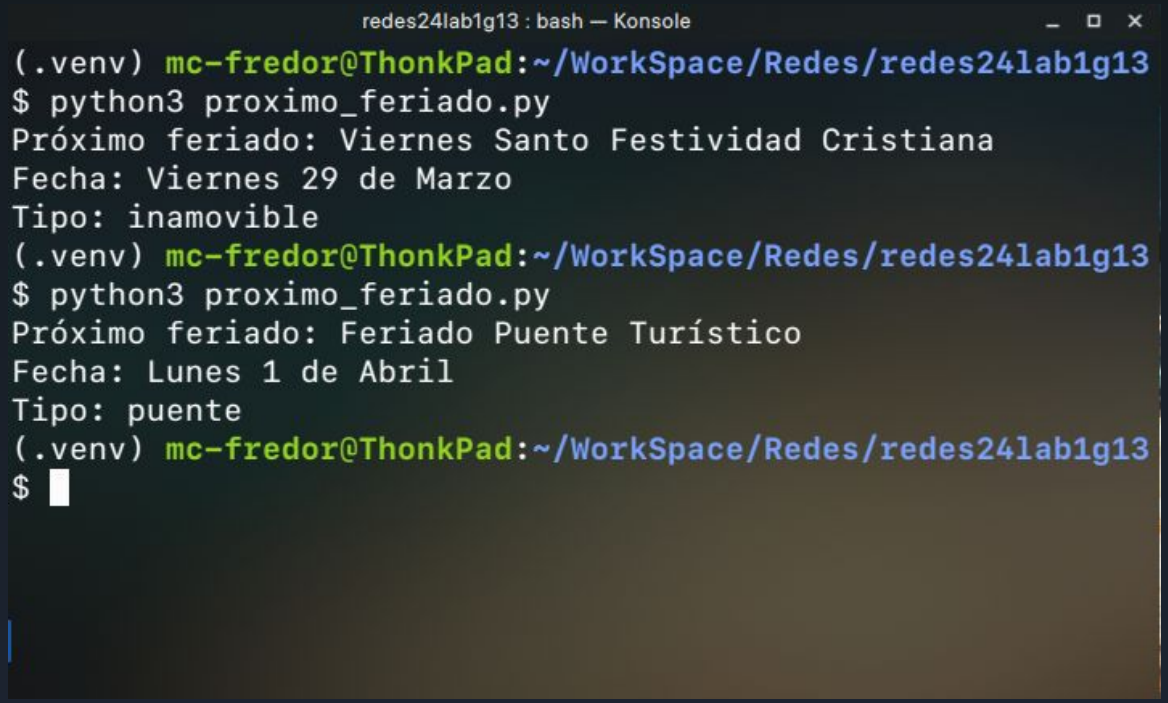
```python
@app.route('/peliculas/<int:id>', methods=['PUT'])
def actualizar_pelicula(id):
    pelicula = next((p for p in peliculas if p['id'] == id), None)
    if pelicula:
        data = request.get_json()
        pelicula.update(data)
        return jsonify(pelicula), 200
    return jsonify({'mensaje': 'Película no encontrada'}), 404


@app.route('/peliculas/<int:id>', methods=['DELETE'])
def eliminar_pelicula(id):
    global peliculas
    peliculas = [p for p in peliculas if p['id'] != id]
    return jsonify({'mensaje': 'Película eliminada'}), 200


@app.route('/peliculas/genero/<string:genero>', methods=['GET'])
def peliculas_por_genero(genero):
    filtradas = [p for p in peliculas if p['genero'].lower() == genero.lower()]
    return jsonify(filtradas), 200
```

En *main.py* ahora tenemos...

```python
@app.route('/peliculas/buscar/<string:query>', methods=['GET'])
def buscar_peliculas(query):
    resultado = [p for p in peliculas if query.lower() in p['titulo'].lower()]
    return jsonify(resultado), 200


@app.route('/peliculas/sugerencia', methods=['GET'])
def sugerir_pelicula():
    pelicula = random.choice(peliculas)
    return jsonify(pelicula), 200


@app.route('/peliculas/sugerencia/<string:genero>', methods=['GET'])
def sugerir_pelicula_por_genero(genero):
    peliculas_genero = [p for p in peliculas if p['genero'].lower() == genero.lower()]
    if peliculas_genero:
        pelicula = random.choice(peliculas_genero)
        return jsonify(pelicula), 200
    return jsonify({'mensaje': 'No se encontraron películas del género solicitado'}), 404


def obtener_nuevo_id():
    if len(peliculas) > 0:
        ultimo_id = peliculas[-1]['id']
        return ultimo_id + 1
    return 1
```

3.
Integración con API de Feriados

En el archivo proximo_feriado.py se utiliza la API de nolaborables.com.ar para obtener información sobre los feriados en Argentina.
La aplicación busca y muestra el próximo feriado disponible.



Nota: Se corrigió el *bug* que cambiaba los días de la semana.

🧐 En la Documentación Oficial nos explica que el método *date.weekday()*:

"Regresa el día de la semana como en un número entero, donde el lunes es 0 y el domingo es 6. Por ejemplo, date(2002, 12, 4).weekday() == 2, un miércoles."

-> Luego se cambia...

```
days = ['Domingo', 'Lunes', 'Martes', 'Miércoles', 'Jueves',
        'Viernes', 'Sábado']
```

-> por...

```
days = ['Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes',
        'Sábado', 'Domingo']
```

->para que funcione correctamente...

```
days[date(year, month, day).weekday()]
```

💜 Se **modifica** el código para que agregar la opción de buscar feriados por tipo:

inamovible | trasladable | nolaborable | puente

-> En *set_next(self, holidays)*:

```
223     # Considera el tipo de feriado en el filtrado
224     filtered_holidays = [
            h for h in holidays if h['tipo'] == self.holiday_type]
            if self.holiday_type else holidays
```

-> Y se define...

```
232     def fetch_holidays(self, holiday_type=None):
233             self.holiday_type = holiday_type  # Almacena el tipo de feriado
234             response = requests.get(get_url(self.year))
235             if response.ok:
236                 data = response.json()
237                 self.set_next(data)
238             else:
239                 print("Error al obtener los datos")
```

En *main.py* ahora tenemos...

```python
@app.route('/recomendar_pelicula_feriado/<string:genero>', methods=['GET'])
def recomendar_pelicula_feriado(genero):
    # Obtiene el próximo feriado
    next_holiday = NextHoliday()
    next_holiday.fetch_holidays()
    if not next_holiday.holiday:
        return jsonify({'mensaje': 'No se encontró el próximo feriado'}), 404

    # Filtra las películas por el género solicitado
    peliculas_genero = [p for p in peliculas if p['genero'].lower() == genero.lower()]
    if not peliculas_genero:
        return jsonify({'mensaje': 'No se encontraron películas del género solicitado'}), 404

    # Selecciona una película al azar del género solicitado
    pelicula_recomendada = random.choice(peliculas_genero)

    # Prepara la respuesta con la información del feriado y la película recomendada
    respuesta = {
        'proximo_feriado': {
            'fecha': f"{next_holiday.holiday['dia']} del {next_holiday.holiday['mes']}",
            'motivo': next_holiday.holiday['motivo']
        },
        'pelicula_recomendada': {
            'titulo': pelicula_recomendada['titulo'],
            'genero': pelicula_recomendada['genero']
        }
    }

    return jsonify(respuesta), 200
```

En la *API de películas* se implementa la *API de feriados* para agregar la siguiente funcionalidad:

Obtener la próxima fecha de feriado y recomendar una película que se ajuste al género solicitado para ese día.

4.
Evaluación de la API:

En *test.py* se agregó:

```python
genero = "Acción"

response = requests.get(f"http://localhost:5000/peliculas/genero/"
                        f"{genero.encode('utf-8').decode('latin-1')}")

if response.status_code == 200:
    peliculas = response.json()
    print(f"Películas existentes del género {genero}")

    for pelicula in peliculas:
        print(f"ID: {pelicula['id']}, Título: {pelicula['titulo']},"
              f"Género: {pelicula['genero']}")
    print()

else:
    print(f"Error al obtener los detalles de las películas de {genero}.")
```

✅

Obtener películas de un género específico.

✅

En *test.py* se agregó:

```
302  response = requests.get(f"http://localhost:5000/peliculas/buscar/"
303                          f"{string.encode('utf-8').decode('latin-1')}")
304
305  if response.status_code == 200:
306      peliculas = response.json()
307      print(f"Se encontraron al menos {len(peliculas)} " +
308            f"coincidencias para {string}")
309
310      for pelicula in peliculas:
311          print(f"ID: {pelicula['id']}, Título: {pelicula['titulo']},"
312                f" Género: {pelicula['genero']}")
313
314      print()
315
316  else:
317      print(f"Error al obtener los detalles de las películas "
318            f"que contienen la string {string}.")
```

✅

Obtener películas que contengan una string específica

string = "ln"

✅

En *test.py* se agregó:

```python
response = requests.get('http://localhost:5000/peliculas/sugerencia')

if response.status_code == 200:
    pelicula = response.json()

    print("Detalles de la película aleatoria:")
    print(f"ID: {pelicula['id']}, Título: {pelicula['titulo']}, "
          f"Género: {pelicula['genero']}")
    print()

else:
    print("Error al obtener los detalles de la película aleatoria.")
```

✅

Sugerir una película random

✅

En *test.py* se agregó:

```
response = requests.get(f"http://localhost:5000/peliculas/sugerencia/"
                        f"{genero.encode('utf-8').decode('latin-1')}")

if response.status_code == 200:
    pelicula = response.json()
    print(f"Detalles de la película aleatoria del genero {genero}:")
    print(f"ID: {pelicula['id']}, Título: {pelicula['titulo']}, "
          f"Género: {pelicula['genero']}")
    print()

else:
    print(f"Error al obtener los detalles de "
          f"la película aleatoria de {genero}.")
```

✅

Sugerir una película random según género

genero = "Acción"

✅

En *test.py* se agregó:

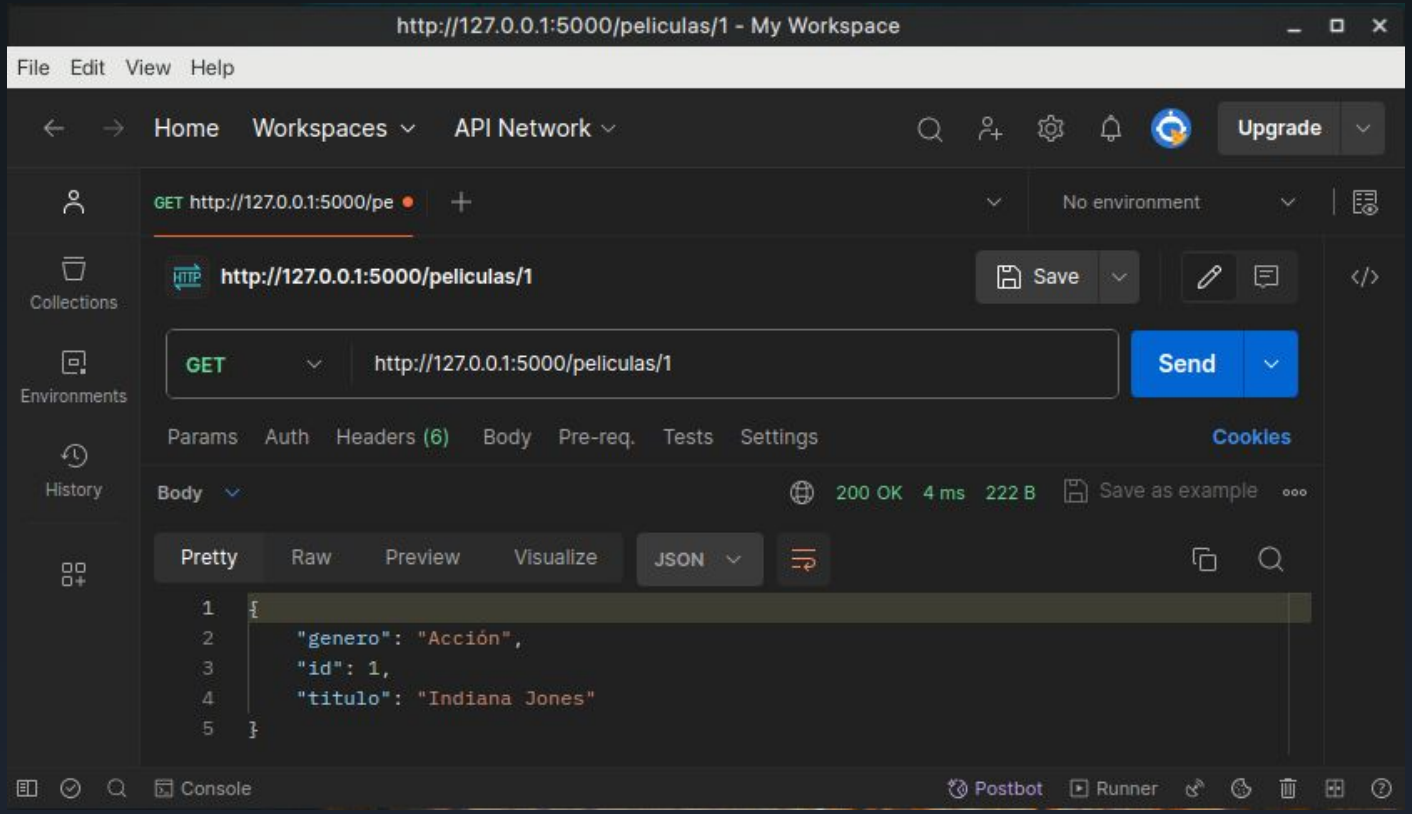
```python
genero = "Acción"
response = requests.get(f"http://localhost:5000/recomendar_pelicula_feriado"
                        f"/{genero.encode('utf-8').decode('latin-1')}")

if response.status_code == 200:
    pelicula = response.json()
    print(f"Detalles de la película aleatoria del genero "
          f"{genero} en el próximo feriado:")
    print(f"ID: {pelicula['pelicula_recomendada']['id']}, "
          f"Título: {pelicula['pelicula_recomendada']['titulo']}, "
          f"Género: {pelicula['pelicula_recomendada']['genero']}, "
          f"Feriado: {pelicula['proximo_feriado']['fecha']} "
          f"- {pelicula['proximo_feriado']['motivo']}")
    print()

else:
    print(f"Error al obtener los detalles de las películas "
          f"de {genero} recomendadas para el próximo feriado.")
```

✅

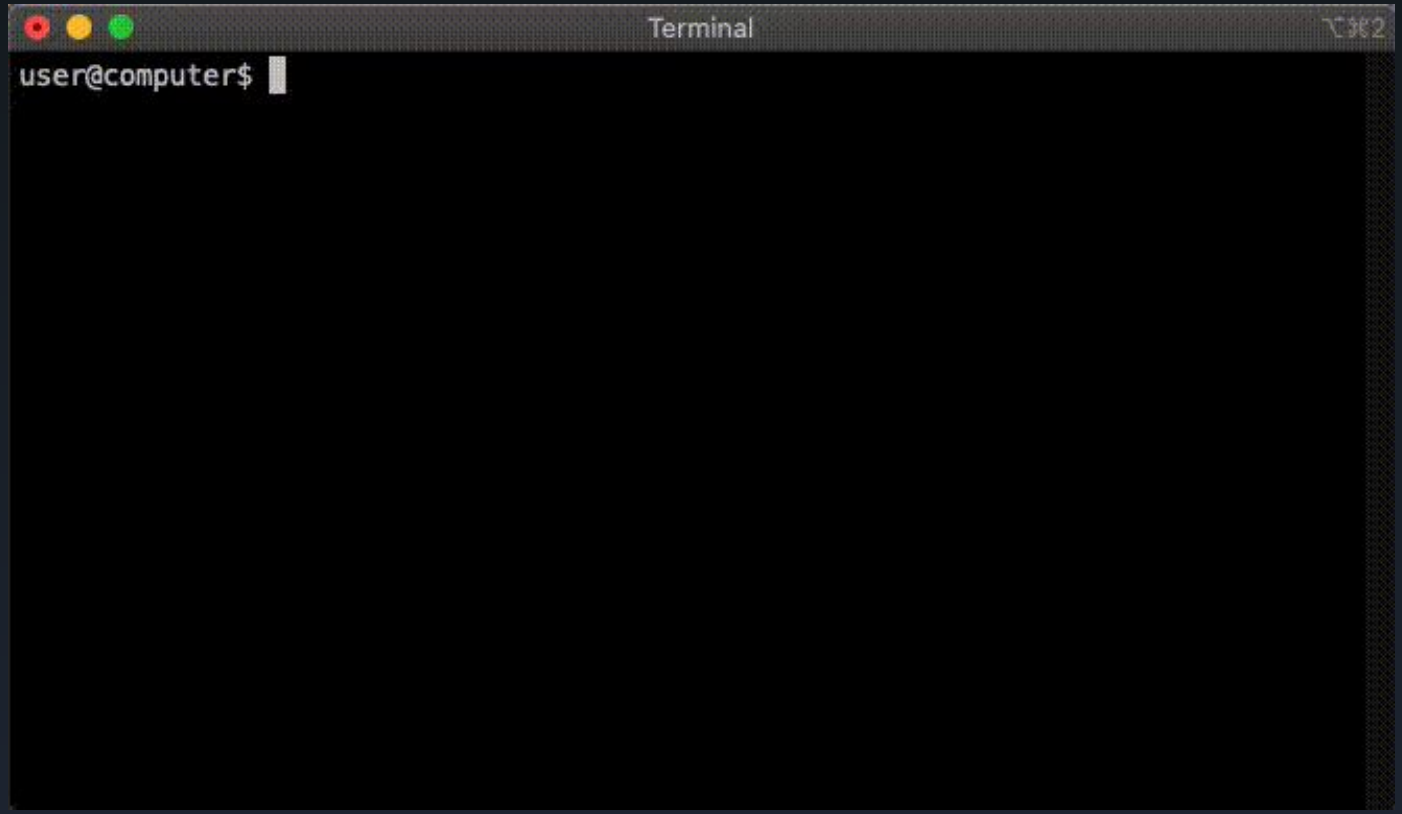
Sugerir una película random según el género para ver en el próximo feriado

✅

Testeo de APIs con Postman



Testeo de APIs con CURL

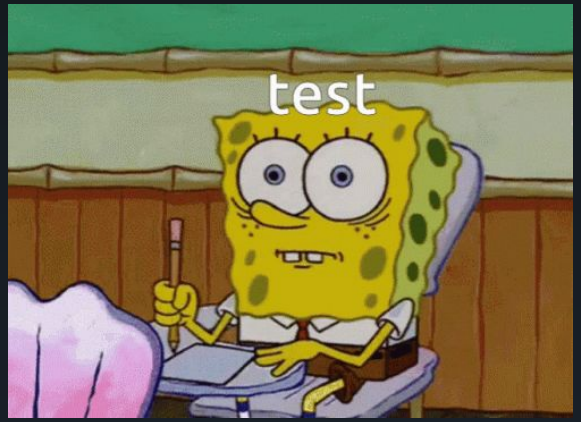


Nota: Se recomienda ver **README.md** para ver el listado completo de *comands-test*, además de pipear estos con el comando **jq** para mayor *legibilidad* al leer la salida.

5.
Conclusiones

Salidas de los *unittest*:

```
(.venv)$ python -m unittest tests/test_app.py
Buscando...
[{'id': 1, 'titulo': 'Indiana Jones - Actualizado', 'genero':
'Acción'}, {'id': 2, 'titulo': 'Star Wars', 'genero':
'Acción'}, {'id': 3, 'titulo': 'Interstellar', 'genero':
'Ciencia ficción'}, {'id': 4, 'titulo': 'Jurassic Park',
'genero': 'Aventura'}, {'id': 5, 'titulo': 'The Avengers',
'genero': 'Acción'}, {'id': 6, 'titulo': 'Back to the
Future', 'genero': 'Ciencia ficción'}, {'id': 7, 'titulo':
'The Lord of the Rings', 'genero': 'Fantasía'}, {'id': 8,
'titulo': 'The Dark Knight', 'genero': 'Acción'}, {'id': 9,
'titulo': 'Inception', 'genero': 'Ciencia ficción'}, {'id':
10, 'titulo': 'The Shawshank Redemption', 'genero': 'Drama'},
{'id': 11, 'titulo': 'Pulp Fiction', 'genero': 'Crimen'},
{'id': 12, 'titulo': 'Fight Club', 'genero': 'Drama'}, {'id':
13, 'titulo': 'Prueba', 'genero': 'Test'}]
.........
----------------------------------------------------------------
--------
Ran 9 tests in 0.044s
```



Salidas de los *unittest*:

```
(.venv)$ python3 test.py
Películas existentes:
ID: 1, Título: Indiana Jones, Género: Acción
ID: 2, Título: Star Wars, Género: Acción
ID: 3, Título: Interstellar, Género: Ciencia ficción
ID: 4, Título: Jurassic Park, Género: Aventura
ID: 5, Título: The Avengers, Género: Acción
ID: 6, Título: Back to the Future, Género: Ciencia ficción
ID: 7, Título: The Lord of the Rings, Género: Fantasía
ID: 8, Título: The Dark Knight, Género: Acción
ID: 9, Título: Inception, Género: Ciencia ficción
ID: 10, Título: The Shawshank Redemption, Género: Drama
ID: 11, Título: Pulp Fiction, Género: Crimen
ID: 12, Título: Fight Club, Género: Drama

Película agregada:
ID: 13, Título: Pelicula de prueba,Género: Acción

Detalles de la película:
ID: 1, Título: Indiana Jones, Género: Acción

Película actualizada:
ID: 1, Título: Nuevo título,Género: Comedia
```

```
Película eliminada correctamente.
Error al obtener los detalles de las películas de Acción.
Se encontraron al menos 3 coincidencias para In
ID: 3, Título: Interstellar, Género: Ciencia ficción
ID: 7, Título: The Lord of the Rings, Género: Fantasía
ID: 9, Título: Inception, Género: Ciencia ficción

Detalles de la película aleatoria:
ID: 6, Título: Back to the Future, Género: Ciencia ficción

Detalles de la película aleatoria del genero Acción:
ID: 8, Título: The Dark Knight, Género: Acción

Detalles de la película aleatoria del genero Acción en el próximo
feriado:
ID: 13,Título: Pelicula de prueba,Género: Acción,Feriado: 29 del 3 -
Viernes Santo Festividad Cristiana - inamovible
```

Salidas de los *test* en curl:

Obtener detalles de una película por ID:


```
{
  "genero": "Acción",
  "id": 1,
  "titulo": "Indiana Jones"
}
```


Actualizar detalles de una película:


```
{
  "genero": "Acción",
  "id": 1,
  "titulo": "Nuevo título"
}
```


Eliminar una película:


```
{
  "mensaje": "Película eliminada"
}
```


Buscar películas por título:


```
[
  {
    "genero": "Acción",
    "id": 2,
    "titulo": "Star Wars"
  }
]
```


Sugerir una película aleatoria:


```
{
  "genero": "Acción",
  "id": 2,
  "titulo": "Star Wars"
}
```


Sugerir una película aleatoria por género:


```
{
  "genero": "Acción",
  "id": 2,
  "titulo": "Star Wars"
}
```


Devolver películas de un género específico:


```
[
  {
    "genero": "Acción",
    "id": 2,
    "titulo": "Star Wars"
  },
  {
    "genero": "Acción",
    "id": 5,
    "titulo": "The Avengers"
  },
  {
    "genero": "Acción",
    "id": 8,
    "titulo": "The Dark Knight"
  }
]
```

¿Qué se puede mejorar?

Nada. Abrazo!

¿Que aprendimos en este lab?

- Como crear un entorno virtual de Python
- Como crear una API con Flask
- Como testear una API y la importancia de esto
- Cómo consumir una API externa
- Cómo utilizar Postman para testear APIs

Muchas gracias por la atención. FIN