REDES Y SISTEMAS DISTRIBUÍDOS

LAB 1

DESARROLLO DE UNA API

Alvaro Medina
Ezequiel Pastore
Lorenzo Canovas
Matías Kühn

CONTEXTO &

📜OBJETIVOS🎯

INTRODUCCIÓN
AL PROYECTO

¿QUÉ ES LO QUE VAMOS A VER?

Desarrollo de una API REST robusta para administrar una base de datos de películas

PYTHON 

FLASK 

“Un código robusto, con buenas decisiones de diseño y documentación para favorecer el mantenimiento y escalabilidad del mismo.”

INTRODUCCIÓN
AL PROYECTO

- Configuración del entorno virtual e instalación de librerías
- Creación de una API con Flask
- Consumo de una API externa
- Evaluación de la API

“Un código robusto, con buenas decisiones de diseño y documentación para favorecer el mantenimiento y escalabilidad del mismo.”

API
REST

CONTENIDOS

Entorno Virtual

Creación API

Consumir API

Testing

Aislamiento de dependencias

Tests de tecnologías de forma controlada

Entorno Virtual

¿Por qué es importante utilizar uno?

Garantía de funcionamiento

Seguridad

Compatibilidad con herramientas modernas

CONFIGURACIÓN

DE UN ENTORNO VIRTUAL

Crear directorio en el cual residirá nuestro proyecto

```
~/Documentos
mkdir redes25lab1g03
```

Ó en caso de disponer de un respositorio, clonarlo

```
~/Documentos
git clone git@bitbucket.org:redes-famaf/redes25lab1g03.git
```

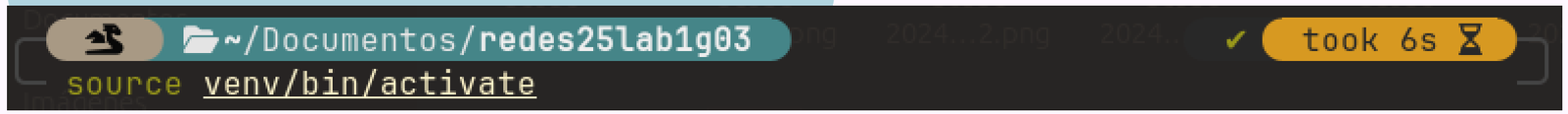
Crear el entorno virtual de Python utilizando venv

```
~/Documentos/redes25lab1g03
python3 -m venv venv
```

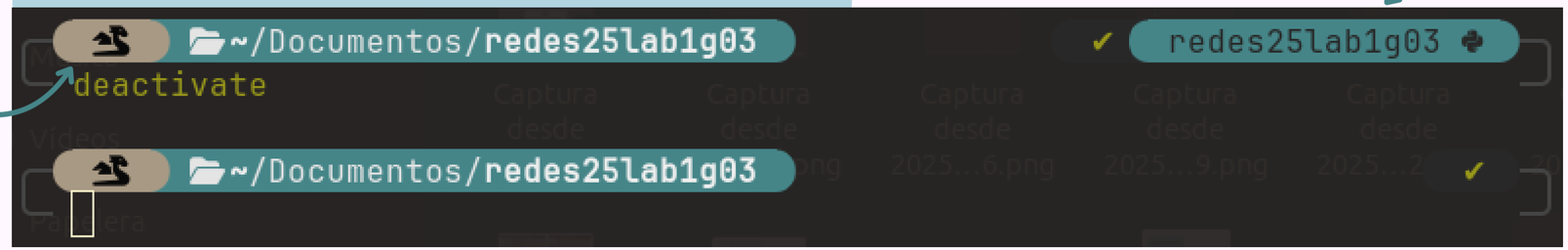
ACTIVACIÓN Y DESACTIVACIÓN

DE UN ENTORNO VIRTUAL

Activar el entorno virtual



Desactivar el entorno virtual



En una terminal promedio, el nombre del entorno suele aparecer de este lado

Podemos ver que el entorno virtual está activado

INSTALACIÓN DE LIBRERÍAS

USANDO PIP

Instalación librería a librería

```
~/Documentos/redes25lab1g03                    redes25lab1g03
pip install flask
```

Instalación de muchas librerías

```
~/Do/Fa/R/🧪/🧪Lab1/redes25lab1g03 on  master
cat requirements.txt
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
```

En un archivo de texto escribimos todas las librerías junto a la versión

Luego de tener las dependencias, instalamos todas las librerías con el comando:

Instalación de muchas librerías

```
~/Documentos/redes25lab1g03                    redes25lab1g03
pip install -r requirements.txt
```

API

DE PELÍCULAS

FUNCIONAMIENTO

manejo y administracion de datos a traves de

CONJUNTO DE FUNCIONALIDADES

interactuar con los datos y el servidor

FLASK

para la creacion y funcionamiento de la aplicacion web y el servidor

LIBRERÍAS

para complementar funcionalidades

veamos en detalle cada cosa

funciones y utilidades

formato json

abstracción

framework

FLASK

brinda un
esquema de trabajo

creacion de url

enrutamiento
de solicitudes

codigos
de solicitudes

FUNCIONALIDADES BÁSICAS DE LA API

RECOMENDAR

+ una película random
+ una película random de un género
+ una película random para el próximo feriado

OBTENER

+ una pelicula en particular
+ todas las películas de un genero
+ obtener todo el catalogo de peliculas

ACTUALIZAR PELÍCULAS

ELIMINAR PELÍCULAS

AGREGAR PELÍCULAS

IMPLEMENTACIÓN
DE LAS FUNCIONALIDADES

FUNCIONAMIENTO

MEDIANTE cURL

PARTES
DE LA API

Nuestra API

Funcionalidades

```python
def obtener_pelicula_random_genero(genero: str) -> Response:
    """
    Obtiene una película random de un género determinado.

    :param str genero: El género de la película por recomendar.

    :returns response pelicula_random: En caso de exito, retornaun objeto JSON
        que contiene la pelicula recomendada del género especificado.
        Si la entrada es vacía, obtenemos un mensaje y un error 404.
        En caso de que no haya películas para recomendar de dicho género,
        la petición tiene éxito y se retorna un objeto JSON vacio.
    """

    # Validar el género
    genero_norm = normalizar_str(genero)

    if genero_norm == "":
        print("Entrada inválida. Por favor, repita la solicitud género válido.")
        return jsonify([]), 404

    # Busco pelicula del género
    list_pelicula = buscar_pelis_genero(genero_norm)

    print(f"List Película ->{list_pelicula}")

    if list_pelicula == []:
        print(
            "No existen películas del genero {genero_norm} entre las disponibles.")
        return jsonify([]), 200

    pelicula_random = random.choice(list_pelicula)

    # Return
    return jsonify(pelicula_random)
```

Metodo add_url_rule

```python
app.add_url_rule('/peliculas', 'obtener_peliculas',
                 obtener_peliculas, methods=['GET'])
app.add_url_rule('/peliculas/<int:id>', 'obtener_pelicula',
                 obtener_pelicula, methods=['GET'])
app.add_url_rule(
    '/peliculas/sugerencia_feriado',
    'obtener_pelicula_sugerida_feriado',
    obtener_pelicula_sugerida_feriado,
    methods=['GET']
)
app.add_url_rule('/peliculas', 'agregar_pelicula',
                 agregar_pelicula, methods=['POST'])
app.add_url_rule('/peliculas/<int:id>', 'actualizar_pelicula',
                 actualizar_pelicula, methods=['PUT'])
app.add_url_rule('/peliculas/<int:id>', 'eliminar_pelicula',
                 eliminar_pelicula, methods=['DELETE'])
app.add_url_rule('/peliculas/<string:genero>', 'obtener_peliculas_mismo_genero',
                 obtener_peliculas_mismo_genero, methods=['GET'])
app.add_url_rule('/peliculas/random/<string:genero>', 'obtener_pelicula_random_genero',
                 obtener_pelicula_random_genero, methods=['GET'])
app.add_url_rule('/peliculas/sugerencia', 'dar_sugerencia',
                 dar_sugerencia, methods=['GET'])

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

Base de datos de Peliculas

redes25lab1g03 > global_var.py > ...

```python
""" Archivo 'global_var.py'
El archivo contiene variables globales muy utilizadas en toda la API
"""

peliculas = [
    {'id': 1, 'titulo': 'Indiana Jones', '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'}
]
generos = [
    "accion",
    "ciencia ficcion",
    "aventura",
    "fantasia",
    "drama",
    "crimen",
    "comedia",
    "suspenso",
    "terror",
    "documental",
    "romance",
    "musical",
    "animacion",
    "guerra"
]

```

¿COMO CONSUMIR UNA API EXTERNA?

👨🏽‍🍳 ¿COMO CONSUMIR UNA API EXTERNA? 🍽️

EN GENERAL 🐧

DOCUMENTACIÓN

Debería **proporcionar los URL** a los diferentes **endpoints** que suministran la información.

En este caso podemos observar la URL a la cual debemos llamar con el método GET (El caso opcional también)

👨🏽‍🍳 ¿COMO CONSUMIR UNA API EXTERNA? 🍽️

EN GENERAL 🐧

DOCUMENTACIÓN

Debería **proporcionar los URL** a los diferentes **endpoints** que suministran la información.

En este caso podemos observar la URL a la cual debemos llamar con el método GET (El caso opcional también)

👨🏽‍🍳 ¿COMO CONSUMIR UNA API EXTERNA? 🍽️

😄API DE FERIADOS📅

Para el laboratorio se hizo uso de una API externa que proporciona los **feriados** que hay en Argentina 🇦🇷

+ Busca y muestra el próximo feriado posible

En el archivo **proximo_feriado.py**

+ Manejamos la lógica para consumir la API

👨🏻‍🍳 ¿COMO CONSUMIR UNA API EXTERNA? 🍽️

proximo_feriado.py

INTEGRACIÓN DE LA API

- PUNTO CLAVE -

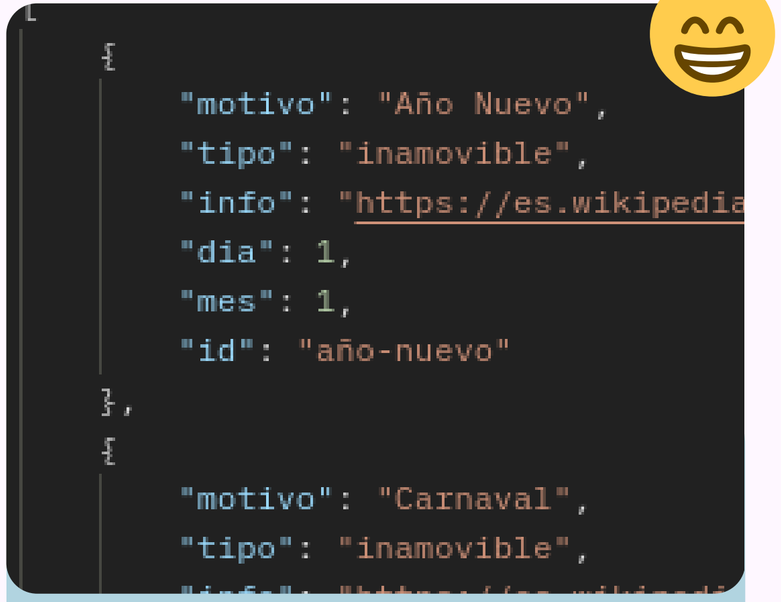
Se define la función **get_url(year)** para construir la URL a donde se solicitarán los feriados del año ingresado.

LUEGO CONSUMIMOS REALIZANDO UNA PETICIÓN GET A LA URL

La función que se encarga de realizar el pedido es **fetch_holidays()**

```
response = requests.get(get_url(self.year))
data = response.json()
```

Ingresamos el año



data = Lista de diccionarios con la información de los **feriados** proporcionada por la API.

EVALUACIÓN DE LA API

🧪 TEST 🧪

HECHOS EN EL LAB

1. **cURL**
2. **test.py**
3. **test_pytest.py**
4. **Postman**
5. **test_pytest_main.py (opcional)**

TEST

HECHOS EN EL LAB

1. **cURL**
2. **test.py**
3. **test_pytest.py**
4. **Postman**
5. **test_pytest_main.py (opcional)**

Algunos métodos HTTP con Curl

Método **GET**

```
$ curl --request GET http://localhost:5000/peliculas/{id}
$ curl --request GET http://localhost:5000/peliculas/random/{genero}
$ curl --request GET http://localhost:5000/peliculas/sugerencia
```

Método **POST & DELETE**

```
$ curl --data "@data.json" --header "Content-Type: application/json" --request POST
https://localhost:5000/peliculas
```

```
$ curl  --request DELETE http://localhost:5000/peliculas/{id}
```

🧪 TEST 🧪

HECHOS EN EL LAB

1. cURL
2. **test.py**
3. **test_pytest.py**
4. **Postman**
5. test_pytest_main.py (opcional)

PRESENTACIÓN Y FUNCIONAMIENTO

test.py

🤔 ¿Qué es pytest y para qué sirven los mocks? 🎭

🤔 ¿Que es pytest y para que sirven los mocks? 🎭

pytest es un **framework de pruebas** para Python que nos permite **automatizar verificaciones** y asegurarnos de que nuestro **código funciona como lo esperamos.**

Los **mocks** nos permiten **simular respuestas de una API** sin necesidad de que el servidor real esté en funcionamiento

🤔 ¿Que es pytest y para que sirven los mocks? 🎭

Esta forma de testear es útil para:

- **Definir el comportamiento esperado de nuestros endpoints**, asegurando que respondan correctamente según las especificaciones.
- Verificar que el **diseño** de la API sea intuitivo y fácil de usar **para los clientes** que la consulten.
- Detectar **inconsistencias entre la implementación simulada y la API real** antes de desplegar el código, evitando errores en producción (esto se puede ver al **desactivar los mocks**).
- Probar **distintos escenarios** (CRUD, gestión de errores, respuestas ante solicitudes incorrectas) sin depender de una conexión real o del estado del servidor.

PRESENTACIÓN Y FUNCIONAMIENTO

test_pytest.py

PRESENTACIÓN Y FUNCIONAMIENTO

test_pytest_main.py

PRESENTACIÓN Y FUNCIONAMIENTO

POSTMAN

Qué diferencias hay entre

test.py

test_pytest.py

postman

test.py

- Servidor en funcionamiento.
- Tests mas sofisticados.
- Chequea que los endpoints implementados trabajen bien.

test_pytest.py

- Simula pruebas y testeos de los endpoints de la API.
- Servidor sin funcionamiento.
- Es nuestra guía.

postman

- Punto de vista del cliente.
- Util para verificar el funcionamiento de nuestros endpoints.

CONCLUSIONES &
PRÓXIMOS PASOS

RESULTADOS
DE LA EVALUACIÓN DE LA API

🔮🚀

POSIBLES MEJORAS

Ó AMPLIACIONES A LA API

POSIBLES MEJORAS

Ó AMPLIACIONES A LA API

AGREGAR MÁS PROPIEDADES A LAS PELÍCULAS

+ Duración
+ Fecha de lanzamiento
+ Rating
+ Links

FILTRAR PELÍCULAS SEGÚN PROPIEDADES

+ Según la duración
+ Según el rating
+ Según la fecha de lanzamiento

DOCUMENTACIÓN DE LA API

Descripción minuciosa de una API para su correcto consumo.

VISUALIZAR LA API MEDIANTE UNA PÁGINA WEB

Implementación de un sitio web que consuma la API.

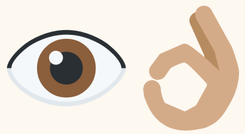
MEJORAS HECHAS

EN LA API

MEJORAS HECHAS

EN LA API

- Manejo de errores
- Modularización de funciones
- Estandarización de las entradas
- Test Unitarios con Postman



PRINCIPALES PUNTOS

PRINCIPALES PUNTOS

Aprendizaje variado:

- API
- Frameworks
- Interacción con servidores
- Consumo de APIs

Experiencia de laboratorio:

- Entretenido
- Claro y conciso
- Sin grandes dificultades
- Intenso

Muchas Gracias