Facultad de Ciencias de la Administración
UNER

TRABAJO INTEGRADOR FINAL

PROGRAMACIÓN 2 - 2023 – 2do cuatrimestre

TECNICATURA UNIVERSITARIA EN DESARROLLO WEB

EL TRABAJO INTEGRADOR FINAL TIENE POR OBJETIVO QUE EL ALUMNO

- Aplique y refuerce los conceptos adquiridos durante toda la cursada.
- Sea capaz de programar un aplicativo mínimo aplicando dichos conocimientos, interpretando y traduciendo correctamente los diagramas de Clases en código Python respetando las relaciones y responsabilidades entre las Clases y los Objetos creados a partir de ellas.

CONDICIONES DE ENTREGA

- El Trabajo Práctico deberá ser:
    - Realizado en forma individual o en **grupos de NO más de 3 (tres) alumnos**.
    - Cargado en la sección del Campus Virtual correspondiente, en un archivo ZIP o RAR con las soluciones a cada ejercicio en código Python. Las soluciones para los ejercicios deben estar contenidas cada una en un archivo .py distinto.
    - En caso de realizar el Trabajo Práctico en grupo, deberá indicarse el apellido y nombre de los integrantes del mismo. Todos los integrantes del grupo deben realizar la entrega en el campus y deberá agregarse al comprimido con las soluciones un archivo *integrantes.txt* con los nombres de los participantes.
    - Entregado antes de la fecha límite informada en el campus.
- El Trabajo Práctico será calificado como Aprobado o Desaprobado.
- Las soluciones del alumno/grupo deben ser de autoría propia. De encontrarse soluciones idénticas entre diferentes grupos, dichos trabajos prácticos serán clasificados como **DESAPROBADO**, lo cual será comunicado en la devolución.

BIBLIOTECA FÍLMICA

Se presenta un repositorio de código que los alumnos deben terminar de codificar. En el mundo de los aplicativos web y móviles, es común encontrar diseños donde los datos están contenidos en algún lugar, al cual los aplicativos acceden consumiendo distintos tipos de servicios web (rest, SOAP, websockets, etc). El código ofrecido refiere a un aplicativo que expone servicios web para realizar consultas sobre la base de datos de una biblioteca fílmica contenida en un archivo JSON.

Los servicios que este aplicativo ofrece son los siguientes:

1. http://localhost:5000/api/actores/<id>
   Se obtiene la información de un actor, sus películas y colegas, como así también los géneros de las películas en las que ha participado.
2. http://localhost:5000/api/actores
   Se consulta el listado completo de actores, además de la cantidad de películas y colegas con los que cuenta dicho actor.
3. http://localhost:5000/api/directores/<id>
   Se obtiene la información de un director, como así también sus películas y géneros de dichas películas.
4. http://localhost:5000/api/directores
   Se consulta el listado completo de directores, como así también las películas y géneros de dichas películas.
5. http://localhost:5000/api/peliculas/<id>
   Se obtiene la información de una película, el nombre del director, género, año de estreno y lista de actores en ella.
6. http://localhost:5000/api/peliculas
   Se consulta el listado completo de películas, y además el nombre del director, género, año de estreno y cantidad de actores en ella.

Los servicios 2., 4. y 6. pueden recibir los parámetros opcionales **orden** y **reverso** (que únicamente puede tomar los valores *si* o *no*), los cuales indiquen el campo por el cual las listas deben ser ordenadas y si el orden debe ser el regular o inverso (ver ejemplos en la sección **Ejecución y Prueba**).

A lo largo de este trabajo los estudiantes deberán codificar los modelos de datos como así también parte de la lógica interna de las consultas expuestas como servicios web. Esto es, en principio, traducir los diagramas de clases a código Python y escribir la lógica que permita recuperar los datos del archivo JSON para ser convertidos a objetos de dichas clases y finalmente ser mostrados al usuario.

El aplicativo está soportado por el framework Flask, por lo que deberá previamente tenerlo instalado para poder correrlo. Para instalarlo corra el siguiente comando por consola:

pip install flask

pip install Flask-RESTful

Para aquellos que corran una versión más avanzada de Python, puede que deban utilizar los siguientes comandos en lugar de los anteriores:

pip3 install flask

pip3 install Flask-RESTFul

De requerirse más detalles sobre la instalación, uso u otras características del framework en cuestión, por favor referirse a la documentación oficial alojada en el siguiente link:

https://flask.palletsprojects.com/en/2.2.x/

Modelado de Datos

El modelo de datos presentado para esta solución está comprendido por la definición de 5 entidades. Estas son los Actores, Directores, las Películas de estos, y finalmente los Géneros de dichas películas. Tanto las clases Actor como Director heredan directamente de la clase Artista.

1. Implementar los siguientes diagramas de clases en los archivos indicados del paquete modelos:

Artista
<<Atributos de clase>> <<Atributos de instancia>> id: str nombre: str
<<Constructores>> Artista(id, nombre: str) <<Comandos>> establecerNombre(nombre: str) <<Consultas>> obtenerId(): str obtenerNombre(): str obtenerGeneros(): Genero[]

   a. Utilizar el archivo *artista.py*
   b. La consulta ***obtenerGeneros*** debe hacer uso del servicio ***obtenerPeliculas*** de las clases que heredan de ella, para obtener la lista de películas asociadas ya sea a un Actor o un Director.

Actor (Artista)
<<Atributos de clase>> <<Atributos de instancia>> id: str nombre: str
<<Constructores>> Actor(id, nombre: str) <<Comandos>> <<Consultas>> obtenerPeliculas(): Pelicula[] obtenerColegas(): Actor[] convertirAJSON(): dict convertirAJSONFull(): dict

a. Utilizar el archivo *actor.py*
b. La consulta ***obtenerPeliculas*** debe hacer uso del servicio ***obtenerPeliculas*** de la clase **Biblioteca** para recuperar todas las películas contenidas en el archivo json. Sobre dicha lista se debe iterar hasta encontrar las películas en las que el actor ha trabajado.
c. La consulta ***obtenerColegas*** debe seguir la misma impronta que el punto anterior para intentar encontrar aquellos actores que han trabajado en la misma película con el actor en cuestión.
d. Se debe sobrecargar el operador de igualdad para que compare el atributo **id** de cada objeto de tipo **Actor**.

Director (Artista)
<<Atributos de clase>> <<Atributos de instancia>>
<<Constructores>> Director(id, nombre: str) <<Comandos>> <<Consultas>> obtenerPeliculas(): Pelicula[] convertirAJSON(): dict convertirAJSONFull(): dict

a. Utilizar el archivo *director.py*
b. La consulta ***obtenerPeliculas*** debe hacer uso del servicio ***obtenerPeliculas*** de la clase **Biblioteca** para recuperar todas las películas contenidas en el archivo json. Sobre dicha lista se debe iterar hasta encontrar las películas en las que el director ha trabajado.
c. Se debe sobrecargar el operador de igualdad para que compare el atributo **id** de cada objeto de tipo **Director**.

Pelicula
<<Atributos de clase>> <<Atributos de instancia>> Id: str nombre: str genero: str director: str actores: str[] anio: int
<<Constructores>> Pelicula(id, nombre, genero, director: str, actores: str[], anio: int) <<Comandos>> establecerNombre(nombre: str) establecerGenero(genero: str) establecerDirector(director: str) establecerActores(actores: str[]) establecerAnio(anio: int) <<Consultas>> obtenerId(): str obtenerNombre: str obtenerGenero: Genero obtenerDirector: Director obtenerActores: Actor[] obtenerAnio: int convertirAJSON(): dict convertirAJSONFull(): dict

a. Utilizar el archivo *pelicula.py*
b. La consulta ***obtenerGenero*** debe hacer uso del servicio ***buscarGenero*** de la clase **Biblioteca** para recuperar el objeto de tipo **Genero** asociado a la película.

c. La consulta ***obtenerDirector*** debe hacer uso del servicio ***buscarDirector*** de la clase **Biblioteca** para recuperar el objeto de tipo **Director** asociado a la película.
d. La consulta **obtenerActores** debe hacer uso del servicio **obtenerActores** de la clase **Biblioteca** para recuperar todos los actores y partiendo de allí encontrar aquellos que trabajaron en la película en cuestión.
e. Se debe sobrecargar el operador de igualdad para que compare el atributo **id** de cada objeto de tipo **Pelicula**.

Genero
<<Atributos de clase>> <<Atributos de instancia>> id: str nombre: str
<<Constructores>> Genero(id, nombre: str) <<Comandos>> establecerNombre(nombre: str) <<Consultas>> obtenerId(): str obtenerNombre(): str convertirAJSON(): dict

a. Utilizar el archivo *genero.py*
b. Se debe sobrecargar el operador de igualdad para que compare el atributo **id** de cada objeto de tipo **Genero**.

Motor de Consultas

Biblioteca es una clase que no posee atributos de instancia. Esto es así porque no sigue el propósito de crear objetos de tipo Biblioteca a partir de ella, sino como entidad para centralizar las consultas a realizarse sobre la base datos. Entonces, tanto las entidades que modelan los objetos del mundo (Artista, Actor, Director, Película, Género) deberán hacer uso de las consultas de Biblioteca directamente sobre la Clase en lugar de un objeto creado a partir de ella.

2. Dada la clase **Biblioteca**, contenida en *biblioteca.py*, y que responde al siguiente diagrama:

Biblioteca
<<Atributos de clase>> archivoDeDatos = "biblioteca.json" actores = Actor[] directores = Director[] generos = Genero[] peliculas = Pelicula[] <<Atributos de instancia>>
<<Constructores>> <<Comandos>> Inicializar() <<Consultas>> obtenerActores(orden, reverso: str): Actor[] obtenerDirectores(orden, reverso: str): Director[] obtenerPeliculas(orden, reverso: str): Pelicula[] obtenerGeneros(orden, reverso: str): Genero[] buscarActor(id: str): Actor buscarDirector(id: str): Director buscarPelicula(id: str): Pelicula buscarGenero(id: str): Genero

a. Utilizar las colecciones *Biblioteca.actores*, *Biblioteca.directores*, *Biblioteca.peliculas*, *Biblioteca.generos* para codificar las consultas según corresponda.
b. Para las consultas ***obtenerActores***, ***obtenerDirectores***, ***obtenerPeliculas*** y ***obtenerGeneros***, si se recibe:
    i. Los parámetros opcionales **orden <> None** y **reverso = True**, devolver una copia de la colección correspondiente, ordenando los objetos de la misma por el atributo indicado en orden inverso.
    ii. Ninguno de los parámetros opcionales, devolver una referencia a la colección correspondiente sin alterar su orden.

c. Para las consultas ***buscarActor***, ***buscarDirector***, ***buscarPelicula*** y ***buscarGenero***:
    i. Utilizar el parámetro formal **id** para navegar la colección y encontrar el elemento que coincida con dicho identificador.
    ii. Si no se encontrase coincidencia, retornar el valor **None**.

d. Codificar la lógica del método interno ***__parsearArchivoDeDatos*** de manera que:
    i. Abra el archivo de datos.
    ii. Cargue la información en una estructura de tipo diccionario.
    iii. Cierre el archivo.
    iv. Retorne una referencia al diccionario anteriormente creado.

e. Codificar la lógica del método interno ***__convertirJsonAListas(listas: dict[])*** de la siguiente manera:
    i. Se espera se completen las 4 colecciones:
        1. *Biblioteca.actores*
        2. *Biblioteca.directores*
        3. *Biblioteca.peliculas*
        4. *Biblioteca.generos*

Ejecución y Prueba

Al finalizar la codificación del trabajo, el alumno podrá verificar que su solución es correcta y cumple con los requisitos para aprobar este trabajo si es que al correr el aplicativo y al visitarse las siguientes URL a través de un navegador web, se obtienen los resultados que se muestra a continuación:

URL: http://127.0.0.1:5000/api/actores/316ae44b-16d8-4490-9be1-016e6415d289

Resultado Esperado:


```
{"nombre": "Leonardo Di Caprio", "generos": ["comedia", "ciencia
ficcion"], "peliculas": [{"nombre": "The Wolf of Wall Street", "anio":
"2013"}, {"nombre": "Inception", "anio": "2010"}], "colegas": ["Margot
Robbie", "Cillian Murphy"]}
```


URL: http://127.0.0.1:5000/api/directores/b8ac5230-107d-45c4-9177-215a199b6b8a

Resultado Esperado:


```
{"nombre": "Christopher Nolan", "generos": ["ciencia ficcion",
"accion"], "peliculas": [{"nombre": "Inception", "anio": "2010"},
{"nombre": "The Dark Knight", "anio": "2008"}]}
```


URL: http://127.0.0.1:5000/api/peliculas/1da2004f-be88-4cd3-93be-1442fbbd78a1
Resultado Esperado:


```
{"nombre": "Barbie", "genero": "comedia", "director": "Greta Gerwig",
"actores": ["Margot Robbie", "Ryan Gosling"], "anio": "2023"}
```


URL: http://127.0.0.1:5000/api/peliculas?orden=nombre&**reverso=si**

Resultado Esperado:


```
[{"nombre": "The Wolf of Wall Street", "genero": "comedia",
"director": "Martin Scorsese", "actores": 2, "anio": "2013"},
{"nombre": "The Dark Knight", "genero": "accion", "director":
"Christopher Nolan", "actores": 2, "anio": "2008"}, {"nombre":
"Inception", "genero": "ciencia ficcion", "director": "Christopher
Nolan", "actores": 2, "anio": "2010"}, {"nombre": "Barbie", "genero":
"comedia", "director": "Greta Gerwig", "actores": 2, "anio": "2023"}]
```

URL: http://127.0.0.1:5000/api/peliculas?orden=nombre&**reverso=no**

Resultado Esperado:


```
[{"nombre": "Barbie", "genero": "comedia", "director": "Greta Gerwig",
"actores": 2, "anio": "2023"}, {"nombre": "Inception", "genero":
"ciencia ficcion", "director": "Christopher Nolan", "actores": 2,
"anio": "2010"}, {"nombre": "The Dark Knight", "genero": "accion",
"director": "Christopher Nolan", "actores": 2, "anio": "2008"},
{"nombre": "The Wolf of Wall Street", "genero": "comedia", "director":
"Martin Scorsese", "actores": 2, "anio": "2013"}]
```