Facultad de Ciencias de la Administración
UNER

TRABAJO INTEGRADOR FINAL

PROGRAMACIÓN 2 - 2022 – 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 4 (cuatro) 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.
- Deben seguirse las convenciones de escritura de código expuestas y consensuadas a partir de la Clase 05.

BIBLIOTECA MUSICAL

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 musical contenida en un archivo JSON.

Los servicios que este aplicativo ofrece son los siguientes:

1. http://localhost:5000/api/artistas/<id>
   Se obtiene la información de un artista, sus álbumes, canciones e integrantes (si correspondiese a una banda).
2. http://localhost:5000/api/artistas
   Se consulta el listado completo de artistas, y además los álbumes, cantidad de canciones e integrantes (si correspondiese a una banda) de cada uno de ellos.
3. http://localhost:5000/api/albumes/<id>
   Se obtiene la información de un álbum, nombre del artista y sus canciones.
4. http://localhost:5000/api/albumes
   Se consulta el listado completo de álbumes, y además nombre de los artistas y las canciones de cada uno de ellos.
5. http://localhost:5000/api/canciones/<id>
   Se obtiene la información de una canción, el nombre del artista y del álbum al que corresponde.
6. http://localhost:5000/api/canciones
   Se consulta el listado completo de canciones, y además el nombre del artista y del álbum al que corresponde cada una de ellas.

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

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 4 entidades. Estas son los Artistas (también llamados músicos), los Álbumes de estos últimos, las Canciones, las cuales están asociadas tanto a los álbumes como a los artistas, y finalmente los Géneros.

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 tipo: str genero: str
<<Constructores>> Artista(id, nombre, tipo, genero: str) <<Comandos>> establecerNombre(nombre: str) establecerTipo(tipo: str) establecerGenero(genero: str) <<Consultas>> obtenerId(): str obtenerNombre(): str obtenerTipo(): str obtenerGenero(): Genero obtenerCanciones(): Cancion[] obtenerAlbumes(): Album[] convertirAJSON(): dict convertirAJSONFull(): dict

   a. Utilizar el archivo *artista.py*

b. La consulta ***obtenerCanciones*** debe hacer uso del servicio ***obtenerCanciones*** de la clase Biblioteca para devolver la lista de canciones que pertenecen al artista.
c. La consulta ***obtenerAlbumes*** debe hacer uso del servicio ***obtenerAlbumes*** de la clase **Biblioteca** para devolver la lista de álbumes que pertenecen al artista.
d. Se debe sobrecargar el operador de igualdad para que compare el atributo id de cada objeto de tipo Artista.

Album
<<Atributos de clase>> <<Atributos de instancia>> id: str nombre: str anio: int genero: str artista: str canciones: str[]
<<Constructores>> Album(id, nombre: str, anio: int, genero, artista: str, canciones: str[]) <<Comandos>> establecerNombre(nombre: str) establecerAnio(anio: int) establecerGenero(genero: str) establecerArtista(artista: str) <<Consultas>> obtenerId(): str obtenerNombre(): str obtenerAnio(): int obtenerGenero(): Genero obtenerArtista(): Artista obtenerCanciones(): Cancion[] convertirAJSON(): dict

convertirAJSONFull(): dict

a. Utilizar el archivo *album.py*
b. La consulta ***obtenerGenero*** debe hacer uso del servicio ***buscarGenero*** de la clase **Biblioteca** para devolver el objeto de tipo **Genero** asociado al álbum.
c. La consulta ***obtenerArtista*** debe hacer uso del servicio ***buscarArtista*** de la clase **Biblioteca** para devolver el objeto de tipo **Artista** asociado al álbum.
d. La consulta ***obtenerCanciones*** debe hacer uso del servicio ***obtenerCanciones*** de la clase **Biblioteca** para devolver la lista de canciones que pertenecen al álbum.
e. Se debe sobrecargar el operador de igualdad para que compare el atributo **id** de cada objeto de tipo **Album**.

Cancion
<<Atributos de clase>> <<Atributos de instancia>> id: str nombre: str artista: str
<<Constructores>> Cancion(id, nombre: str, artista: str) <<Comandos>> establecerNombre(nombre: str) establecerArtista(artista: str) <<Consultas>> obtenerId(): str obtenerNombre(): str obtenerArtista(): Artista obtenerAlbum(): Album convertirAJSON(): dict

a. Utilizar el archivo *cancion.py*
b. La consulta ***obtenerArtista*** debe hacer uso del servicio ***buscarArtista*** de la clase **Biblioteca** para devolver el objeto de tipo **Artista** asociado al álbum.
c. La consulta ***obtenerAlbum*** debe hacer uso del servicio ***obtenerAlbumes*** de la clase **Biblioteca** para devolver el objeto de tipo **Album** asociado a la canción.
d. Se debe sobrecargar el operador de igualdad para que compare el atributo **id** de cada objeto de tipo **Cancion**.

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**.

Banda
<<Atributos de clase>> <<Atributos de instancia>> Integrantes: str[]
<<Constructores>> Banda(id, nombre, tipo, genero: str, integrantes: str[]) <<Comandos>> establecerIntegrantes(integrantes: str[]) <<Consultas>> obtenerIntegrantes(): str[]

a. Crear el archivo *banda.py*
b. Se deben heredar atributos y comportamiento de la clase **Artista**.

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 (Álbum, Artista, Banda, Canción, 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" artistas = Artista[] canciones = Cancion[] albumes = Album[] generos = Genero[] <<Atributos de instancia>>
<<Constructores>> <<Comandos>> Inicializar() <<Consultas>> obtenerArtistas(orden, reverso: str): Artista[] obtenerCanciones(orden, reverso: str): Cancion[] obtenerAlbumes(orden, reverso: str): Album[] obtenerGeneros(orden, reverso: str): Genero[] buscarArtista(id: str): Artista buscarCancion(id: str): Cancion buscarAlbum(id: str): Album buscarGenero(id: str): Genero

a. Utilizar las colecciones *Biblioteca.artistas*, *Biblioteca.canciones*, *Biblioteca.artistas*, *Biblioteca.generos* para codificar las consultas según corresponda.
b. Para las consultas ***obtenerArtistas***, ***obtenerCanciones***, ***obtenerAlbumes*** 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 ***buscarArtista***, ***buscarCancion***, ***buscarAlbum*** 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.artistas*
        2. *Biblioteca.canciones*
        3. *Biblioteca.albumes*
        4. *Biblioteca.generos*
    ii. *Biblioteca.artistas* puede contener objetos de tipo Artista o de tipo Banda, dependiendo del valor que tome el atributo “tipo” de cada elemento de la base de datos de artistas.

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://localhost:5000/api/artistas/b8ac5230-107d-45c4-9177-215a199b6b8a

Resultado Esperado:


```
{"nombre": "Babasonicos", "tipo": "banda", "genero": "pop", "albumes":
[{"nombre": "Trinchera", "anio": "2022"}], "canciones": ["Mimos son
Mimos", "Paradoja", "Bye Bye", "Vacio", "Anubis", "La Izquierda de la
Noche", "Mentira Nordica", "Madera Ideologica", "Viento y Marea",
"Capital Afectivo", "Lujo"], "integrantes": [{"nombre": "Adrian
Dargelos Rodriguez", "instrumentos": "Voz principal"}, {"nombre":
"Diego Uma Rodriguez", "instrumentos": "Guitarras, coros y
percusion"}, {"nombre": "Diego Uma-T Tunon", "instrumentos":
"Teclados"}, {"nombre": "Diego Panza Castellanos", "instrumentos":
"Bateria"}, {"nombre": "Mariano Roger Dominguez", "instrumentos":
"Guitarras y voz"}]}
```


URL: http://localhost:5000/api/albumes/1da2004f-be88-4cd3-93be-1442fbbd78a1

Resultado Esperado:


```
{"nombre": "Trinchera", "genero": "pop", "anio": "2022", "artista":
"Babasonicos", "canciones": ["Mimos son Mimos", "Paradoja", "Bye Bye",
"Vacio", "Anubis", "La Izquierda de la Noche", "Mentira Nordica",
"Madera Ideologica", "Viento y Marea", "Capital Afectivo", "Lujo"]}
```


URL: http://localhost:5000/api/canciones/5219b4f9-6205-43a8-91ca-c22c7effad48
Resultado Esperado:


```
{"nombre": "Mimos son Mimos", "artista": "Babasonicos", "album":
"Trinchera"}
```


URL: http://localhost:5000/api/albumes?orden=anio&**reverso=no**

Resultado Esperado:


```
[{"nombre": "Joanne", "genero": "pop", "anio": "2016"}, {"nombre":
"Trinchera", "genero": "pop", "anio": "2022"}, {"nombre": "Temporada
de Reggaeton 2", "genero": "trap", "anio": "2022"}]
```

URL: http://localhost:5000/api/albumes?orden=anio&**reverso=si**

Resultado Esperado:


```
[{"nombre": "Trinchera", "genero": "pop", "anio": "2022"}, {"nombre":
"Temporada de Reggaeton 2", "genero": "trap", "anio": "2022"},
{"nombre": "Joanne", "genero": "pop", "anio": "2016"}]
```