IIC3103 - Taller de Integración
Departamento Ciencia de la Computación
Escuela de Ingeniería
Pontificia Universidad Católica

Enunciado **Tarea 2**

Objetivo

El objetivo de esta tarea es crear una API REST que acepte *requests* GET, POST, PUT y DELETE.

Trabajo a realizar

Deben exponer una API que permita **crear/obtener/eliminar** ligas**,** equipos y jugadores, y una vez creados, **entrenar** a los jugadores. Estas acciones se llevarán a cabo a través de distintos requests HTTP a su API.

Descripción general

Para la realización de esta tarea deben implementar una API, la cual debe contar con una base de datos en la que se almacenarán tres modelos: uno que represente a las ligas de deportes, uno que represente a los equipos que compiten en las ligas y otro que represente a los jugadores. La descripción detallada de estos modelos puede encontrarse en el anexo 1.

Finalmente, deben exponer una API REST que permita interactuar con los elementos en su base de datos. Las requests a implementar se detallan en el siguiente apartado.

Requests a implementar

A continuación, se presentan los *requests* que su API debe ser capaz de soportar.

Descripción endpoints

Crear:

- **POST /leagues**: crea una liga y retorna la liga.
- **POST /leagues/<league_id>/teams**: crea un equipo de la liga <league_id> y retorna el equipo creado.
- **POST /teams/<team_id>/players**: crea un jugador del equipo <team_id> y retorna el jugador creado.

Obtener:

- **GET /leagues**: retorna todas las ligas.
- **GET /teams**: retorna todos los equipos.
- **GET /players**: retorna todos los jugadores.
- **GET /leagues/<league_id>**: retorna la liga <league_id>.
- **GET /leagues/<league_id>/teams**: retorna todos los equipos de la liga <league_id>.
- **GET /leagues/<league_id>/players**: retorna todos los jugadores de la liga <league_id>.
- **GET /teams/<team_id>**: retorna el equipo <team_id>.
- **GET /teams/<team_id>/players**: retorna todos los jugadores del equipo <team_id>.
- **GET /players/<player_id>**: retorna el jugador <player_id>.

Entrenar:

- **PUT /leagues/<league_id>/teams/train**: entrena a todos los jugadores de todos los equipos de la liga <league_id>.
- **PUT /teams/<team_id>/players/train**: entrena a todos los jugadores del equipo <team_id>.
- **PUT /players/<player_id>/train**: entrena al jugador <player_id>.

Eliminar (CASCADE):

- **DELETE /leagues/<league_id>**: elimina la liga <league_id> y todos sus equipos.

- **DELETE /teams/<team_id>**: elimina el equipo <team_id> y todos sus jugadores.
- **DELETE /players/<player_id>**: elimina al jugador <player_id

Descripción endpoints

En la siguiente URL encontrará la documentación completa de la API a implementar, incluyendo *requests* y *responses*, y principales códigos de error:

> **URL API**
>
> https://app.swaggerhub.com/apis-docs/Integracion-2021-2/integracionify-api/2.0.0

Además, en las siguientes URLs podrá encontrar la documentación en formato OpenAPI 3.0:

> **OpenAPI 3.0 formato JSON**
>
> https://api.swaggerhub.com/apis/Integracion-2021-2/integracionify-api/2.0.0
>
> **OpenAPI 3.0 formato YAML**
>
> https://api.swaggerhub.com/apis/Integracion-2021-2/integracionify-api/2.0.0/swagger.yaml

Identificador único

Los identificadores **<league_id>, <team_id>** y **<player_id>** construyen utilizando otros atributos de cada elemento (detalle de los modelos en anexo 1)**.** Estarán codificados utilizando el esquema Base64[1] y deben seguir[2] el siguiente formato:

Para obtener el identificador de una liga **<league_id>**, debemos codificar el string **"<league_name>:<league_sport>"**, y quedarnos con los primeros 22 caracteres.[3]

Luego, para obtener el identificador de un equipo **<team_id>**, debemos codificar el string **"<team_name>:<team_city>"**, y quedarnos con los primeros 22 caracteres.

Finalmente, para obtener el identificador de un jugador **<player_id>**, debemos codificar el string **"<player_name>:<player_position>"**, y quedarnos con los primeros 22 caracteres.

En el anexo 2 podrán encontrar ejemplos de esta generación de id's paso a paso que pueden usar de guía, además de algunas librerías de Javascript, Python y Ruby recomendadas.

Arquitectura de la solución

La API podrá estar construida sobre un framework de desarrollo de API's o un framework MVC tal como Rails (Ruby), Django (Python), Sails (Nodejs) u otros. No hay requerimientos específicos sobre lenguajes o frameworks a utilizar.

La solución deberá tener una capa de almacenamiento de datos, con persistencia mínima de 1 hora, tiempo suficiente para soportar la ejecución de varios flujos en serie, esto quiere decir que se deberá implementar una base de datos u otro sistema de almacenamiento que permita ir almacenando los registros que se van creando en los flujos que permite la API.

Se recomienda hacer un deploy en Heroku, sistema de servidores Cloud que tiene una capa gratuita con capacidades suficientes para el desarrollo de esta tarea.

Versionamiento del código

El sitio y todo su código fuente deberá estar versionado en un repositorio de Github Classroom proporcionado por el equipo docente.

---

[1] https://developer.mozilla.org/es/docs/Glossary/Base64
[2] Al corregir la tarea (consumir la API) se va a verificar que se estén creando correctamente los identificadores.
[3] Los identificadores siempre tendrán largo <= 22 caracteres.

Entregables

Cada alumno deberá entregar, mediante un formulario publicado en el sitio del curso, las siguientes *url's*:

- URL del endpoint que permite consumir la api
- URL del repositorio Github.

Adicionalmente, cada estudiante deberá inscribir su url en un formulario que se habilitará para testing de la API.

Para la tarea, cada alumno deberá crear un repositorio en Github Classrooms, en el siguiente link:

> **Creación repositorio Github Classroom**
>
> https://classroom.github.com/a/FBZDIsyr

Fecha de entrega

La tarea se deberá entregar el día 1 de octubre antes de las 18 hrs.

Requisitos mínimos

Las tareas que no cumplan las siguientes condiciones no serán corregidas y serán evaluados con la nota mínima:

- La API deberá ser pública, accesible desde cualquier dispositivo conectado a internet.
- El código deberá estar versionado en su totalidad en un repositorio Git
- El sitio debe reflejar fielmente el código entregado en el repositorio
    - Se recomienda configurar un deploy automático en Heroku asociado al repositorio Github

Penalizaciones

Se descontarán 0,2 puntos de la nota de la tarea por cada hora de atraso en la entrega, contados a partir de la fecha estipulada en el punto anterior.

Cualquier intento de copia, plagio o acto deshonesto en el desarrollo de la tarea, será penalizado con nota 1,1 de acuerdo a la política de integridad académica del DCC.

Anexo 1: Modelos de la base de datos

Cada **liga** debe tener los siguientes atributos:

- ID. (string)
- Name. (string)
- Sport. (string)
- Teams. (url)[4]
- Players. (url)[5]
- Self. (url)[6]

Cada **equipo** debe tener los siguientes atributos:

- ID. (string)
- Name. (string)
- City. (string)
- League. (url)
- Players. (url)
- Self. (url)

Finalmente, cada **jugador** debe tener los siguientes atributos:

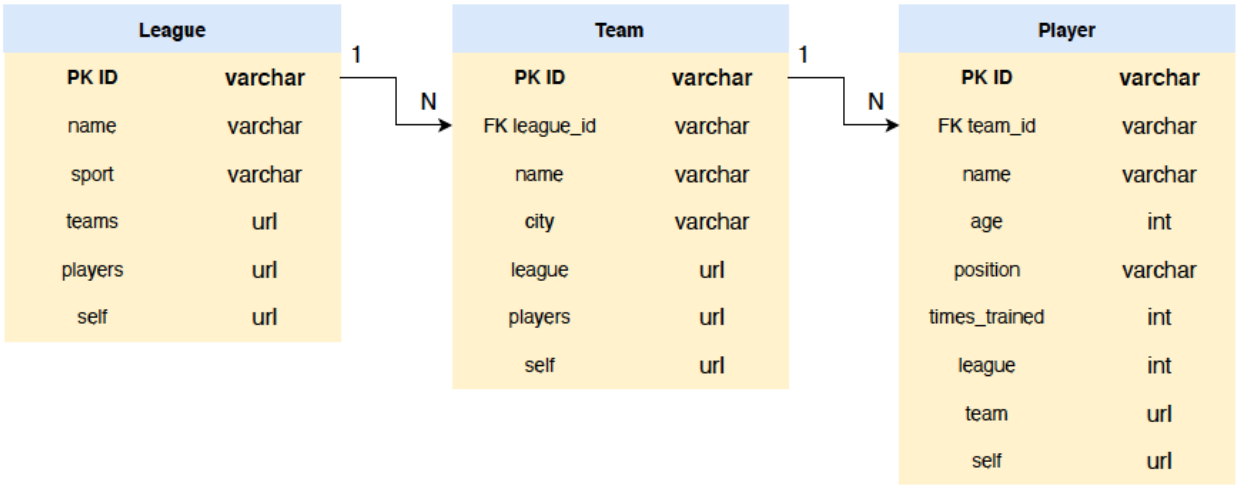
- ID. (string)
- Name. (string)
- Age. (int)
- Position. (string)
- Times Trained. (int)
- League. (url)
- Team. (url)
- Self. (url)

Tengan en cuenta que no es **estrictamente necesario** definir todos estos campos en la base de datos, por ejemplo las URLs podrían ser generadas al serializar el objeto.

Respecto a las relaciones entre estos modelos, cada liga puede tener N equipos, pero cada equipo puede tener solo 1 liga. De la misma forma, cada equipo puede tener N jugadores, pero cada jugador puede pertenecer a 1 solo equipo. La modelación de la base de datos puede verse en el siguiente diagrama:

---

[4] Referencia a una url en la misma API. En este caso, sería apuntando al endpoint: **/leagues/<league_id>/teams**.
[5] Referencia a una url en la misma API. En este caso, sería apuntando al endpoint: **/leagues/<league_id>/players**.
[6] Referencia hacía sí mismo. En el caso de la liga, sería apuntando al endpoint **/leagues/<league_id>.**


League
PK ID varchar
name varchar
sport varchar
teams url
players url
self url
1
N
Team
PK ID varchar
FK league_id varchar
name varchar
city varchar
league url
players url
self url
1
N
Player
PK ID varchar
FK team_id varchar
name varchar
age int
position varchar
times_trained int
league int
team url
self url

Anexo 2: Ejemplos de codificación[7] en Base64

Ejemplo 1:

- **Liga**:
  - **Nombre**: "NBA"
  - **Deporte**: "Basketball"
  - **Identificador** = encode(**"NBA:Basketball"**) = **"TkJBOkJhc2tldGJhbGw="**
- **Equipo**:
  - **Nombre**: "Chicago Bulls"
  - **Ciudad:** "Chicago"
  - **Identificador** = encode(**"Chicago Bulls:Chicago"**)
    = **"Q2hpY2FnbyBCdWxsczpDaG"**
- **Jugador**:
  - **Nombre**: "Michael Jordan"
  - **Position:** "Escolta"
  - **Identificador** = encode(**"Michael Jordan:Escolta"**)
    = **"TWljaGFlbCBKb3JkYW46RX"**

Ejemplo 2:

- **Liga**:
  - **Nombre**: "MLB"
  - **Deporte:** "Baseball"
  - **Identificador** = encode(**"MLB:Baseball"**) = **"TUxCOkJhc2ViYWxs"**
- **Equipo**:
  - **Nombre**: "New York Yankees"
  - **Ciudad:** "New York"
  - **Identificador** = encode(**"New York Yankees:New York"**)
    = **"TmV3IFlvcmsgWWFua2Vlcz"**
- **Jugador**:
  - **Nombre**: "Babe Ruth"
  - **Position:** "Pitcher"
  - **Identificador** = encode(**"Babe Ruth:Pitcher"**)
    = **"QmFiZSBSdXRoOlBpdGNoZX"**

[7] En los ejemplos la función encode(string) ya retorna los primeros 22 caracteres del string codificado en Base64.

Librerías Recomendadas:

```
let string = "Premier League"
let encoded = btoa(string)
console.log(encoded)

> UHJlbWllciBMZWFndWU=
```


```
from base64 import b64encode
string = "Premier League"
encoded = b64encode(string.encode()).decode('utf-8')
print(encoded)

> UHJlbWllciBMZWFndWU=
```

```
require "base64"
string = "Premier League"
encoded = Base64.encode64(string)
puts encoded

> UHJlbWllciBMZWFndWU=
```

*Notar que aquí no se están truncando los primeros 22 caracteres del resultado.

Otra herramienta que les puede ser útil: https://coding.tools/base64-encode

Anexo 3: Recursos adicionales

A continuación, se presentan una serie de links que le podrán ser útiles para el desarrollo de la tarea.

	Ruby on Rails	Python Django	Node
Cursos y documentación	Instalar: https://gorails.com/setup/osx/10.15-catalina Documentación: https://guides.rubyonrails.org/getting_started.html Curso: https://www.codecademy.com/learn/learn-rails	https://docs.djangoproject.com/en/3.0/intro/tutorial01/ https://docs.docker.com/compose/django/[8] ***	Tutorial: https://itnext.io/a-new-and-better-mvc-pattern-for-node-express-478a95b09155
Despliegue de una app en Heroku	https://devcenter.heroku.com/articles/getting-started-with-rails5#store-your-app-in-git	https://devcenter.heroku.com/articles/django-app-configuration	https://devcenter.heroku.com/articles/deploying-nodejs

*** Este link es muy útil, pero está desactualizado en un detalle. En el paso 2 de “Connecting to the database”, es necesario agregar la siguiente línea: `'PASSWORD': 'postgres'`.

[8] Tutorial para crear un proyecto utilizando Django, Docker y Postgres.

Resolviendo dudas

Comportamiento de la API

Al **crear** exitosamente una liga/equipo/jugador, deben retornar status HTTP **201 (Created).**

Al **obtener/entrenar** una liga/equipo/jugador, deben retornar status HTTP **200 (Ok).**

Al **eliminar** una liga/equipo/jugador, deben retornar status HTTP **204 (No Content).**

Las condiciones para **crear exitosamente** una liga/equipo/jugador son las siguientes:

- No debe existir una liga/equipo/jugador con el mismo identificador único. Si ya existe, deben retornar el existente, con status HTTP **409 (Conflict)**.
- Deben venir todos los campos necesarios, cumpliendo con los tipos de datos requeridos. Si falta un campo o el tipo del campo es incorrecto, como por ejemplo que se envíe un *int* en vez de *string*, deben retornar status HTTP **400 (Bad Request)**.
- Al crear un equipo, la liga debe existir anteriormente. En caso de no existir, deben retornar status HTTP **422 (Unprocessable Entity)**. Lo mismo aplica al crear jugadores.

La única condición para **obtener, entrenar o eliminar exitosamente** una liga/equipo/jugador es la siguiente:

- Debe existir una liga/equipo/jugador con el identificador señalado en la url. En caso de no existir, deben retornar status HTTP **404 (Not Found)**.

En todos los endpoints, ante cualquier otro request con un método que no estén esperando, deben retornar status HTTP **405 (Method Not Allowed)**

Identificadores Únicos

Los tildes pueden traerles algunos problemas al codificar, no se preocupen de esto ya que al evaluar no se intentará crear liga/equipo/jugador con tildes en sus nombres.

Por otro lado, no se preocupen de las “colisiones” entre los identificadores de objetos distintos, no se va a testear ningún caso borde que provoque ese tipo de problemas.

StackOverflow

Foro más popular para resolver dudas sobre código, *frameworks* y lenguajes de programación. Antes de preguntar, asegúrate que la pregunta no se ha realizado anteriormente.

> http://stackoverflow.com/

Resolución de dudas en Github

También se resolverán dudas mediante los *issues* del repositorio en github:
https://github.com/IIC3103/2021-2-dudas/issues