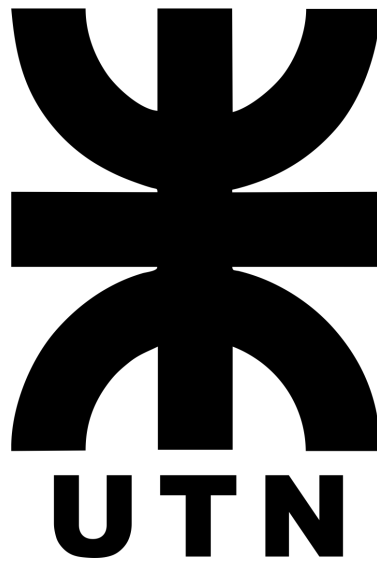


# **UNIVERSIDAD TECNOLÓGICA NACIONAL**



**REGIONAL CÓRDOBA**

**SECRETARIA DE ASUNTOS ESTUDIANTILES  
2025**

**TALLER DE DESARROLLO C#**

**PROFESOR:**

- Genaro Rafael Bergesio

**COORDINADOR:**

- Octavio Felix Cavalleris Malanca

**SECRETARIO SAE:**

- Exequiel Carranza

# Índice

<b>Introducción.....</b>	<b>1</b>
<b>Explicación de Funcionalidades.....</b>	<b>4</b>
Entrenadores.....	4
Descripción.....	4
Tabla.....	5
Torneos.....	5
Descripción.....	5
Tablas.....	6
Gimnasios.....	6
Descripción.....	6
Tablas.....	7
Items.....	8
Descripción.....	8
Tablas.....	9
Ciudades.....	9
Descripción.....	9
Tablas.....	10
Ligas.....	10
Descripción.....	10
Tablas.....	11
Hall of Fame.....	12
Descripción.....	12
Tablas.....	13
<b>Consideraciones.....</b>	<b>14</b>
Documentación.....	14
Entrega.....	14
Evaluación.....	14
No Obligatoriedad.....	15
Inscripción.....	15

# Introducción

Aunque desde el curso maneje los contenidos de manera progresiva para que los estudiantes pudieran seguir a su ritmo el desarrollo de aplicativos C# considero que la mejor forma de reforzar el conocimiento es que se realice un trabajo de manera propia. Por este motivo, y para emitir un certificado de participación, realizó esta actividad práctica orientada a la creación de módulos propios que cumplan funcionalidades en las aplicaciones ya creadas.

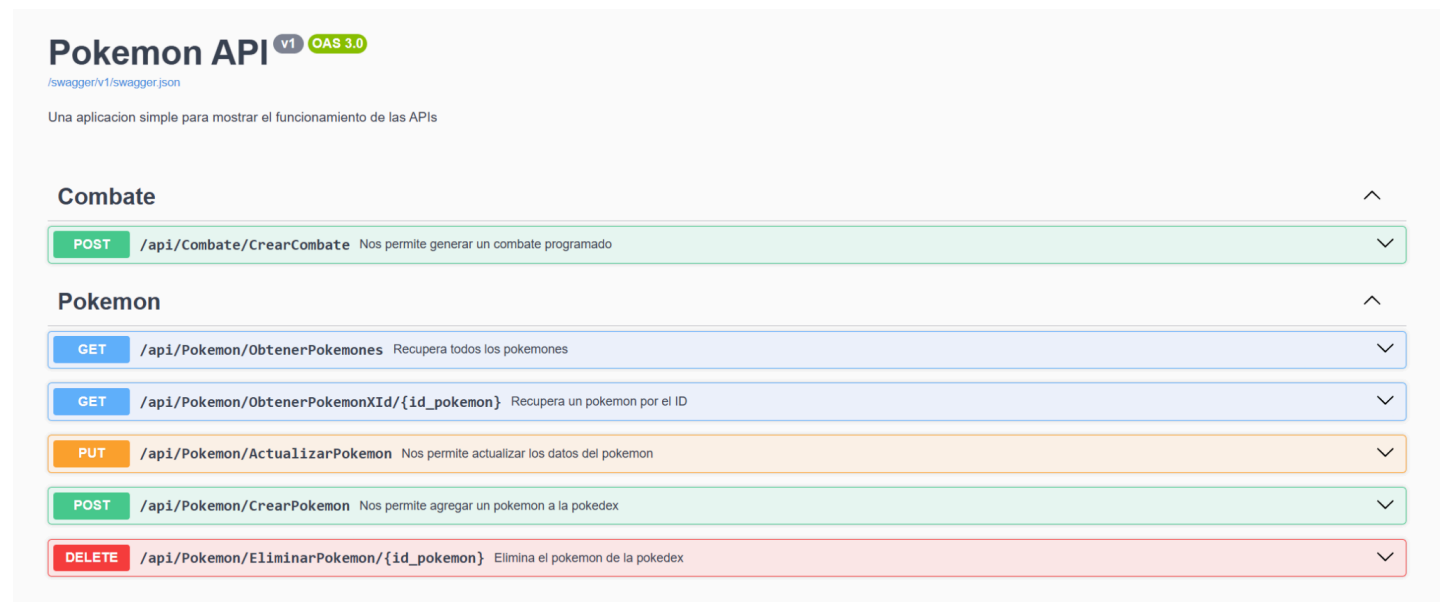
Cada grupo deberá elegir una funcionalidad descrita en este PDF, anotar su grupo en la planilla correspondiente y crear desde cero la estructura de las bases de datos, el controlador con sus endpoints y la funcionalidad en la aplicación de escritorio para probarlo. Para facilitar la implementación y no tener que conceder permisos individualmente a cada estudiante, la idea es que ustedes creen su propio repositorio en base a los provistos por el curso y los compartan con el docente para su corrección.

Los repositorios en cuestión son:

<https://github.com/83464-Bergesio-Genaro/PokeAPICurso>

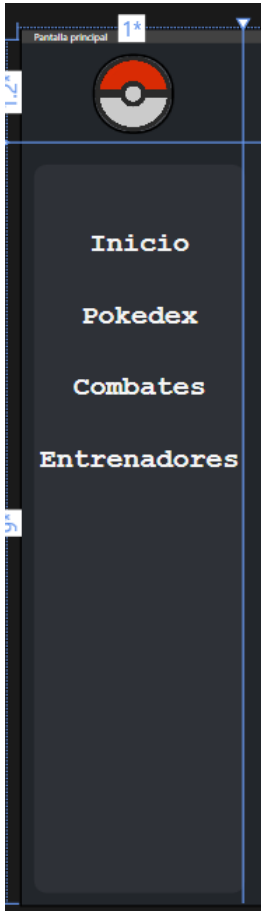
<https://github.com/83464-Bergesio-Genaro/PokeDexCurso>

En la API los estudiantes deberán desarrollar un controlador y ser visualizado mediante swagger:



Cada endpoints solicitado debe estar correctamente documentado siguiendo los lineamientos planteados en clase, teniendo en consideración los diferentes códigos de respuesta.

Mientras que en la aplicación de escritorio deberá ser accesible mediante el menú lateral izquierda de la aplicación:



Está demás aclarar pero el funcionamiento de la aplicación de escritorio debe estar alineada con el de la API, es decir que debe consumir los endpoints desarrollado por ustedes.

La cantidad de los integrantes de grupos debe estar entre 5 a 8 los cuales se pueden dividir las tareas de desarrollo de las aplicaciones como deseen. La fecha de entrega será aproximadamente el 20 de agosto pero puede ser **modificable** dependiendo los avances de los grupos en este trabajo. La idea es que **cada grupo tenga una funcionalidad** diferente, en caso que se completen todas las funcionalidades se va a permitir repetir. Para que se entienda, cada grupo elige una funcionalidad anotando sus integrantes en el google sheets.

Les dejo el listado de funcionalidades:

1. Entrenadores
2. Torneos
3. Gimnasios
4. Items
5. Ciudades
6. Liga Pokemon (Campeon, elite 4)
7. Hall of Fame

Cada funcionalidad tendrá su conjunto de reglas de negocios que deben seguir para su desarrollo, además les voy a dejar las tablas que deben tener. Si no pueden crear las mismas me pueden mandar un email para que lo resolvamos.

Probablemente para muchos de las tablas que van a desarrollar necesiten la tabla Regiones que tiene este formato:

id	INT IDENTITY(0,1)
nombre	VARCHAR(255)
generacion_aparicion	INT

```
CREATE TABLE [dbo].[Regiones](
    [id] [int] IDENTITY(0,1) NOT NULL,
    [nombre] [varchar](255) NOT NULL,
    [generacion_aparicion] [int] NOT NULL,
    CONSTRAINT [PK_Regiones] PRIMARY KEY CLUSTERED
(
    [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO
```

Con los siguientes registros:

```
SET IDENTITY_INSERT [dbo].[Regiones] ON

INSERT [dbo].[Regiones] ([id], [nombre], [generacion_aparicion]) VALUES (0,
N'Kanto', 1)
INSERT [dbo].[Regiones] ([id], [nombre], [generacion_aparicion]) VALUES (1,
N'Jotho', 2)
INSERT [dbo].[Regiones] ([id], [nombre], [generacion_aparicion]) VALUES (2,
N'Hoen', 3)
INSERT [dbo].[Regiones] ([id], [nombre], [generacion_aparicion]) VALUES (3,
N'Sinnoh', 4)
INSERT [dbo].[Regiones] ([id], [nombre], [generacion_aparicion]) VALUES (4,
N'Teselia', 5)
SET IDENTITY_INSERT [dbo].[Regiones] OFF
```

# Explicación de Funcionalidades

## Entrenadores

### Descripción

Desarrollar un controlador el cual permita dar de alta los diferentes entrenadores que existan en las regiones. Los mismos deben tener los datos: ID, nombres, apellidos, fecha nacimiento, entrenador activo y cantidad de medallas. También debe ser posible modificarlos, consultarlos y darlos de baja lógicamente (atributo entrenador activo).

Los endpoints deben ser lo siguiente:

- (GET) **Entrenadores/api/ObtenerEntrenadoresCompleto**
  - Recupera todos los entrenadores no importa si están activos o no
  - Codigos: 200, 204, 400, 409, 500
- (GET) **Entrenadores/api/ObtenerEntrenadoresActivo**
  - Recupera solo los entrenadores activos
  - Codigos: 200, 204, 400, 409, 500
- (GET) **Entrenadores/api/ObtenerEntrenadorXid/{id\_entrenador}**
  - Recupera un entrenador en base a su identificador único.
  - Codigos: 200, 204, 400, 409, 500
- (POST) **Entrenadores/api/CargarEntrenador**
  - En el body se deben enviar todos los datos del entrenador y debe devolver el resultado en caso que sea exitoso.
  - Códigos: 201, 400, 409, 500
- (PUT) **Entrenadores/api/ModificarEntrenador**
  - En el body se deben enviar todos los datos del entrenador para modificarlos y devolver el mismo si fue correcta la modificación. Si no encuentra ese registro utilicen el código 404.
  - Codigos: 200, 400, 404, 409, 500
- (PATCH/DELETE) **Entrenadores/api/RetirarEntrenador/{id\_entrenador}**
  - Utilizando el PathParam deben enviar el id de aquel entrenador que quieren que no se encuentre más activo.
  - Codigos: 200, 400, 404, 409, 500

En la aplicación de escritorio deben desarrollar un User Control como el que utilice en pokemon donde carguen un listado con los datos de los entrenadores. Esa pantalla tiene que tener la posibilidad de ver un listado de todos los entrenadores, otro de únicamente los activos, una opción para cargar un nuevo entrenador, otra para modificar uno que esté activo y la posibilidad de retirarlo.

En la misma APP deben respetar los colores, fuentes y estilos que se están utilizando, pueden desarrollar propios si también lo desean siguiendo más o menos la paleta de colores existente.

## Tabla

Nombre Atributo	Tipo
id <b>(PK)</b>	INT, debe tener IDENTITY(0,1)
nombres	VARCHAR(255)
apellidos	VARCHAR(255)
fecha_nacimiento	DATE
entrenador_activo	BIT
cantidad_medallas	INT

## Torneos

### Descripción

Desarrollar un controlador el cual permita dar de alta los diferentes torneos que existan en las regiones. Los mismos deben tener los datos: id, nombre del torneo, id region, fecha inicio, fecha fin, torneo activo y mínimo medallas. También debe ser posible modificarlos, consultarlos y darlos de baja lógicamente (atributo torneo activo).

Los endpoints deben ser lo siguiente:

- (GET) **Torneos/api/ObtenerTorneoCompleto**
  - Recupera todos los torneos no importa si están activos o no
  - Codigos: 200, 204, 400, 409, 500
- (GET) **Torneos/api/ObtenerTorneosActivo**
  - Recupera solo los torneos activos
  - Codigos: 200, 204, 400, 409, 500
- (GET) **Torneos/api/ObtenerTorneoXid/{id\_torneo}**
  - Recupera un torneo en base a su identificador único.
  - Codigos: 200, 204, 400, 409, 500
- (POST) **Torneos/api/CargarTorneo**
  - En el body se deben enviar todos los datos del torneo y debe devolver el resultado en caso que sea exitoso.
  - Códigos: 201, 400, 409, 500
- (PUT) **Torneos/api/ModificarTorneo**

- En el body se deben enviar todos los datos del torneo para modificarlos y devolver el mismo si fue correcta la modificación. Si no encuentra ese registro utilice el código 404.
- Codigos: 200, 400, 404, 409, 500
- (PATCH/DELETE) **Torneos/api/DesactivarTorneo/{id\_torneo}**
  - Utilizando el PathParam deben enviar el id del torneo que quieren que no se encuentre más activo.
  - Codigos: 200, 400, 404, 409, 500

En la aplicación de escritorio deben desarrollar un User Control como el que utilice en pokemon donde carguen un listado con los torneos disponibles. Esa pantalla tiene que tener la posibilidad de ver un listado de todos los torneos activos, una opción para cargar un nuevo torneo, otra para modificarlo y la posibilidad de desactivarlo.

En la misma APP deben respetar los colores, fuentes y estilos que se están utilizando, pueden desarrollar propios si también lo desean siguiendo más o menos la paleta de colores existente.

## Tablas

Nombre Atributo	Tipo
id <b>(PK)</b>	INT, debe tener IDENTITY(0,1)
nombre_torneo	VARCHAR(255)
id_region <b>(FK)</b>	INT
fecha_inicio	DATE
fecha_fin	DATE
torneo_activo	BIT
minimo_medallas	INT

## Gimnasios

### Descripción

Desarrollar un controlador el cual permita dar de alta los diferentes gimnasios que existan en las regiones. Los mismos deben tener los datos: id, nombre de la ciudad donde está ubicado, id region, fecha de creación del gimnasio, nombre del líder, gimnasio activo y el nombre de la medalla. También debe ser posible modificarlos, consultarlos y darlos de baja lógicamente (atributo gimnasio activo).

Los endpoints deben ser lo siguiente:



- (GET) **Gimnasios/api/ObtenerGimnasiosCompleto**
  - Recupera todos los gimnasios no importa si están activos o no
  - Codigos: 200, 204, 400, 409, 500
- (GET) **Gimnasios/api/ObtenerGimnasiosActivo**
  - Recupera solo los gimnasios activos
  - Codigos: 200, 204, 400, 409, 500
- (GET) **Gimnasios/api/ObtenerGimnasioXid/{id\_gimnasio}**
  - Recupera un gimnasio en base a su identificador único.
  - Codigos: 200, 204, 400, 409, 500
- (POST) **Gimnasios/api/CargarGimnasio**
  - En el body se deben enviar todos los datos del gimnasio y debe devolver el resultado en caso que sea exitoso.
  - Códigos: 201, 400, 409, 500
- (PUT) **Gimnasios/api/ModificarGimnasio**
  - En el body se deben enviar todos los datos del gimnasio para modificarlos y devolver el mismo si fue correcta la modificación. Si no encuentra ese registro utilice el código 404.
  - Codigos: 200, 400, 404, 409, 500
- (PATCH/DELETE) **Gimnasios/api/DesactivarGimnasio/{id\_gimnasio}**
  - Utilizando el PathParam deben enviar el id del gimnasio que quieren que no se encuentre más activo.
  - Codigos: 200, 400, 404, 409, 500

En la aplicación de escritorio deben desarrollar un User Control como el que utilice en pokemon donde carguen un listado con los gimnasios disponibles. Esa pantalla tiene que tener la posibilidad de ver un listado de todos los gimnasios activos, una opción para cargar uno nuevo, otra para modificar y la posibilidad de desactivarlo.

En la misma APP deben respetar los colores, fuentes y estilos que se están utilizando, pueden desarrollar propios si también lo desean siguiendo más o menos la paleta de colores existente.

## Tablas

Nombre Atributo	Tipo
id (PK)	INT, debe tener IDENTITY(0,1)
ciudad_gimnasio	VARCHAR(255)
id_region (FK)	INT
fecha_alta	DATE
entrenador_lider	VARCHAR(255)
gimnasio_activo	BIT

nombre_medalla	VARCHAR(255)
----------------	--------------

## Items

### Descripción

Desarrollar un controlador el cual permita dar de alta los diferentes ítems que existan y los usuarios puedan utilizar. Los mismos deben tener los datos: id, nombre del ítem, stock máximo, fecha de creación en el sistema, efecto o descripción y si el ítem está activo. También debe ser posible modificarlos, consultarlos y darlos de baja lógicamente (atributo item activo).

Los endpoints deben ser lo siguiente:

- (GET) **Items/api/ObtenerItemsCompleto**
  - Recupera todos los items no importa si están activos o no
  - Codigos: 200, 204, 400, 409, 500
- (GET) **Items/api/ObtenerItemsActivo**
  - Recupera solo los ítems activos
  - Codigos: 200, 204, 400, 409, 500
- (GET) **Items/api/ObtenerItemXid/{id\_item}**
  - Recupera un gimnasio en base a su identificador único.
  - Codigos: 200, 204, 400, 409, 500
- (POST) **Items/api/CargarItem**
  - En el body se deben enviar todos los datos del ítem y debe devolver el resultado en caso que sea exitoso.
  - Códigos: 201, 400, 409, 500
- (PUT) **Items/api/ModificarItem**
  - En el body se deben enviar todos los datos del ítem para modificarlos y devolver el mismo si fue correcta la modificación. Si no encuentra ese registro utilice el código 404.
  - Codigos: 200, 400, 404, 409, 500
- (PATCH/DELETE) **Items/api/DesactivarItem/{id\_item}**
  - Utilizando el PathParam deben enviar el id del ítem que quieren que no se encuentre más activo.
  - Codigos: 200, 400, 404, 409, 500

En la aplicación de escritorio deben desarrollar un User Control como el que utilice en pokemon donde carguen un listado con los ítems disponibles. Esa pantalla tiene que tener la posibilidad de ver un listado de todos los ítems activos, una opción para cargar uno nuevo, otra para modificar y la posibilidad de desactivarlo.

En la misma APP deben respetar los colores, fuentes y estilos que se están utilizando, pueden desarrollar propios si también lo desean siguiendo más o menos la paleta de colores existente.

## Tablas

Nombre Atributo	Tipo
id <b>(PK)</b>	INT, debe tener IDENTITY(0,1)
nombre_item	VARCHAR(255)
stock_maximo	INT
fecha_creacion	DATE
efecto	VARCHAR(255)
item_activo	BIT

## Ciudades

### Descripción

Desarrollar un controlador el cual permita dar de alta las diferentes ciudades que existan en las regiones. Las mismas deben tener los datos: id, nombre de la ciudad, cantidad habitantes, fecha de creación de la ciudad, descripción de la ciudad, si tiene gimnasio activo y a que región pertenece. También debe ser posible modificarlos, consultarlos y darlos de baja lógicamente (atributo tiene gimnasio).

Los endpoints deben ser lo siguiente:

- (GET) **Ciudades/api/ObtenerCiudadesCompleto**
  - Recupera todas las ciudades que se encuentren cargadas
  - Codigos: 200, 204, 400, 409, 500
- (GET) **Ciudades/api/ObtenerCiudadesGimnasio**
  - Recupera solo las ciudades que posean un gimnasio activo
  - Codigos: 200, 204, 400, 409, 500
- (GET) **Ciudades/api/ObtenerCiudadXid/{id\_ciudad}**
  - Recupera una ciudad mediante su identificador unico
  - Codigos: 200, 204, 400, 409, 500
- (POST) **Ciudades/api/CargarCiudad**
  - En el body se deben enviar todos los datos de la ciudad y debe devolver el resultado en caso que sea exitoso.
  - Códigos: 201, 400, 409, 500
- (PUT) **Ciudades/api/ModificarCiudad**
  - En el body se deben enviar todos los datos de la ciudad para modificarlos y devolver la misma si fue correcta la modificación. Si no encuentra ese registro utilice el código 404.
  - Codigos: 200, 400, 404, 409, 500

- (PATCH/DELETE) **Ciudades/api/DeshabilitarGimnasio/{id\_ciudad}**
  - Utilizando el PathParam deben enviar el id de la ciudad que quieren que se desactive su gimnasio.
  - Codigos: 200, 400, 404, 409, 500

En la aplicación de escritorio deben desarrollar un User Control como el que utilice en pokemon donde carguen un listado con las ciudades disponibles. Esa pantalla tiene que tener la posibilidad de ver un listado de todas las ciudades con gimnasio, una opción para cargar una nueva, otra para modificar y la posibilidad de deshabilitar su gimnasio.

En la misma APP deben respetar los colores, fuentes y estilos que se están utilizando, pueden desarrollar propios si también lo desean siguiendo más o menos la paleta de colores existente.

## Tablas

Nombre Atributo	Tipo
id (PK)	INT, debe tener IDENTITY(0,1)
nombre_ciudad	VARCHAR(255)
cantidad_habitantes	INT
fecha_creacion	DATE
descripcion_ciudad	VARCHAR(255)
tiene_gimnasio	BIT
id_region (FK)	INT

## Ligas

### Descripción

Desarrollar un controlador el cual permita dar de alta las diferentes ligas pokémon que existan en las regiones. Las mismas deben tener los datos: id, nombre del campeón, los nombres de los entrenadores de la elite 4, fecha de creación de la liga, si la liga está activa y a que región pertenece. En caso que una liga se comparta entre regiones poner la primera región en la que apareció. También debe ser posible modificarlos, consultarlos y darlos de baja lógicamente (atributo item activo).

Los endpoints deben ser lo siguiente:

- (GET) **Ligas/api/ObtenerLigasCompleto**
  - Recupera todas las ligas que se encuentren cargadas
  - Codigos: 200, 204, 400, 409, 500

- (GET) **Ligas/api/ObtenerLigasActivas**
  - Recupera solo las ligas que estén activas actualmente
  - Codigos: 200, 204, 400, 409, 500
- (GET) **Ligas/api/ObtenerLigasXid/{id\_liga}**
  - Recupera una liga mediante su identificador único
  - Codigos: 200, 204, 400, 409, 500
- (POST) **Ligas/api/CargarLiga**
  - En el body se deben enviar todos los datos de la liga y debe devolver el resultado en caso que sea exitoso.
  - Códigos: 201, 400, 409, 500
- (PUT) **Ligas/api/ModificarLiga**
  - En el body se deben enviar todos los datos de la liga para modificarlos y devolver la misma si fue correcta la modificación. Si no encuentra ese registro utilice el código 404.
  - Codigos: 200, 400, 404, 409, 500
- (PATCH/DELETE) **Ciudades/api/DeshabilitarLiga/{id\_liga}**
  - Utilizando el PathParam deben enviar el id de la liga que quieren que no se encuentre más activo.
  - Codigos: 200, 400, 404, 409, 500

En la aplicación de escritorio deben desarrollar un User Control como el que utilice en pokemon donde carguen un listado con las ligas disponibles. Esa pantalla tiene que tener la posibilidad de ver un listado de todas las ligas activas, una opción para cargar una nueva, otra para modificar y la posibilidad de dar de baja lógicamente la liga.

En la misma APP deben respetar los colores, fuentes y estilos que se están utilizando, pueden desarrollar propios si también lo desean siguiendo más o menos la paleta de colores existente.

## Tablas

Nombre Atributo	Tipo
id <b>(PK)</b>	INT, debe tener IDENTITY(0,1)
nombre_campeon	VARCHAR(255)
nombre_elite_1	VARCHAR(255)
nombre_elite_2	VARCHAR(255)
nombre_elite_3	VARCHAR(255)
nombre_elite_4	VARCHAR(255)
liga_activa	BIT
fecha_creacion	Date

id_region (FK)	INT
----------------	-----

## Hall of Fame

### Descripción

Desarrollar un controlador el cual permita dar de alta los diferentes entrenadores que han logrado ganar la liga pokémon. De cada registro se necesita tener: id, nombre del entrenador que venció la liga, los id de los pokemons que utilizó, momento de cuándo ocurrió (Fecha y hora), si el entrenador es el campeón actual y a que región pertenece. En caso que una liga se comparta entre regiones poner la primera región en la que apareció. También debe ser posible modificarlos, consultarlos y deshabilitar si no es más el campeón de la liga (Atributo campeón activo).

Los endpoints deben ser lo siguiente:

- (GET) **HallFame/api/ObtenerHallFameCompleto**
  - Recupera a todos los entrenadores que ganaron la liga.
  - Codigos: 200, 204, 400, 409, 500
- (GET) **HallFame/api/ObtenerHallFameCampeones**
  - Recupera solo aquellos entrenadores que ganaron la liga y son actualmente los campeones a vencer.
  - Codigos: 200, 204, 400, 409, 500
- (GET) **HallFame/api/ObtenerHallFameXid/{id\_hall}**
  - Recupera un ganador por su id.
  - Codigos: 200, 204, 400, 409, 500
- (POST) **HallFame/api/CargarGanador**
  - En el body se deben enviar todos los datos del entrenador y pokemons con los cuales ganó la liga.
  - Códigos: 201, 400, 409, 500
- (PUT) **HallFame/api/ModificarGanador**
  - En el body se deben enviar todos los datos para modificarlos y devolver la misma si fue correcta la modificación. Si no encuentra ese registro utilice el código 404.
  - Codigos: 200, 400, 404, 409, 500
- (PATCH/DELETE) **Hall/api/DeshabilitarGanador/{id\_liga}**
  - Utilizando el PathParam deben enviar el id del registro del hall of fame de aquel entrenador que ya no es más el campeón a vencer.
  - Codigos: 200, 400, 404, 409, 500

En la aplicación de escritorio deben desarrollar un User Control como el que utilice en pokemon donde carguen un listado con todos los entrenadores que ganaron la liga Esa pantalla tiene que tener la posibilidad de ver un listado de todos los entrenadores que son campeones, una opción

para cargar un nuevo registro del HoF, otro para modificarlo y la posibilidad de “retirar” a un campeón de su actividad.

En la misma APP deben respetar los colores, fuentes y estilos que se están utilizando, pueden desarrollar propios si también lo desean siguiendo más o menos la paleta de colores existente.

## Tablas

Nombre Atributo	Tipo
id <b>(PK)</b>	INT, debe tener IDENTITY(0,1)
nombre_entrenador	VARCHAR(255)
id_primer_pokemon	INT
id_segundo_pokemon	INT (Puede ser NULL)
id_tercer_pokemon	INT (Puede ser NULL)
id_cuarto_pokemon	INT (Puede ser NULL)
id_quinto_pokemon	INT (Puede ser NULL)
id_sexta_pokemon	INT (Puede ser NULL)
campeon_activo	BIT
fecha_hora_ocurrencia	Datetime(0)
id_region <b>(FK)</b>	INT

# Consideraciones

## Documentación

Como se menciona en la introducción es importante que todos los endpoints de la API estén debidamente comentados con sus posibles respuestas. Además a nivel código las funciones utilizadas también deben contener una explicación.

En la parte de escritorio no es necesario tanto detalle pero sí dejar explicado aquellas funciones que resultaron más importantes para el desarrollo del programa

## Entrega

No es necesario realizar **ningún tipo de informe** para acompañar las aplicaciones pero si es necesario que al menos un integrante del grupo envíe por vía email el link del repositorio que van a estar utilizando. Pueden utilizar las ramas que consideren necesarias y notificar acerca de cuál es la última versión para así poder revisar el código.

## Evaluación

Para mantenerlo simple y no tener problemas con las base MDF implementadas en el IIS lo que voy a realizar es descargar las versiones informadas por cada grupo, levantarlas en tiempo de ejecución y realizar las pruebas pertinentes a las aplicaciones. Es importante que la aplicación de escritorio esté apuntando correctamente a la API, que va a estar en localhost en el puerto que ustedes le digan.

En caso que los programas no cumplan con los requisitos mínimos puedo solicitar una nueva entrega para que esté aprobado. Los puntos a evaluar son:

### 1. API REST

- a. Cumplimiento de los nombres de los endpoints
- b. Documentación adecuada
- c. Legibilidad del código
- d. Funcionalidad correcta. (Es decir, que si hago un POST cargue un nuevo registro en la Base)

### 2. WPF

- a. Integración correcta con el código original
- b. Documentación adecuada
- c. Legibilidad del código
- d. Implementación con API
- e. Funcionalidad correcta. Lo mismo que antes, que la aplicación realice las operaciones correctamente.




Si considero que el trabajo es lo suficientemente correcto les voy a estar comunicando vía email el resultado e informando a la secretaría de asuntos estudiantiles para expedir los certificados en el futuro.

## No Obligatoriedad

Sobre todo les recuerdo que este trabajo práctico **NO** es obligatorio para seguir en el curso, las clases se van a desarrollar exactamente igual y si ustedes no desean ningún tipo de certificado no va haber ningún problema. También si por motivos personales no quieren entregar el trabajo práctico porque están rindiendo finales (y/o otros motivos) pero en el futuro quieren algún tipo de feedback de aplicaciones que hagan en base a este trabajo práctico no duden en enviarme un email.

## Inscripción

Les dejo el link del google sheets que voy a utilizar para que ustedes se carguen en los grupos, mi recomendación es que intenten llegar al mínimo de integrantes por grupo (5). También va a estar subido en la UV.

 Grupos TPI Curso

Cualquier inconveniente que tengan avisenme.