UNIVERSIDAD NEBRIJA

ALUMNO:

EPS

Asignatura: Arquitectura y Programación de sistemas en Internet			
Curso: 2025/2026	Examen:	Final	Fecha: 18-12-2025
Semestre: Primer	Convocatoria: Ordinaria		

Examen: Desarrollo de una API GraphQL – Pokémon

Contexto

Se desea desarrollar una **API GraphQL** para gestionar un pequeño sistema de Pokémon, entrenadores y capturas.
La API permitirá a los entrenadores iniciar su aventura, autenticarse, consultar Pokémon disponibles y gestionar los Pokémon que han capturado.

Para ello, se proporciona el siguiente **esquema GraphQL**, que deberá ser implementado utilizando **Apollo Server** y **Node.js**.

UNIVERSIDAD NEBRIJA

Esquema proporcionado

```
enum PokemonType {
    NORMAL
    FIRE
    WATER
    ELECTRIC
    GRASS
    ICE
    FIGHTING
    POISON
    GROUND
    FLYING
    PSYCHIC
    BUG
    ROCK
    GHOST
    DRAGON
}

type Pokemon {
    _id: ID!
    name: String!
    description: String!
    height: Float!
    weight: Float!
    types: [PokemonType!]!
}

type OwnedPokemon {
    _id: ID!
     #En base datos se guardará solo el id, encadenado pokemon.
    pokemon: Pokemon!
    nickname: String
    attack: Int!
    defense: Int!
    speed: Int!
    special: Int!
    level: Int!
}
```

```
type Trainer {
    _id: ID!
    name: String!
     #En base datos se guardará solo el id, encadenado
OwnedPokemon.
    pokemons: [OwnedPokemon]!
}

type Mutation {
    startJourney(name: String!, password: String!): String!
    login(name: String!, password: String!): String!
    createPokemon(
        name: String!,
        description: String!,
        height: Float!,
        weight: Float!,
        types: [PokemonType!]!
    ): Pokemon!
    catchPokemon(pokemonId: ID!, nickname: String):
OwnedPokemon!
    freePokemon(ownedPokemonId: ID!): Trainer!
}

type Query {
    me: Trainer
    pokemons(page: Int, size: Int): [Pokemon]!
    pokemon(id: ID!): Pokemon
}
```

Objetivo del examen

Implementar una API GraphQL completamente funcional que cumpla con el esquema anterior, incluyendo **resolvers**, **autenticación** y **gestión de datos**.

UNIVERSIDAD NEBRIJA

Requisitos funcionales

1. Autenticación

- `startJourney`:
  - Crea un nuevo entrenador con nombre y contraseña.
  - Devuelve un **token de autenticación** (por ejemplo, JWT).
  - No se permite crear dos entrenadores con el mismo nombre.
  - Devuelve un token de autenticación.
- `login`:
  - Autentica a un entrenador existente.
  - Devuelve un token si las credenciales son correctas.
  - En caso contrario, lanza un error.

2. Pokémon

- `createPokemon`: (Con autenticación)
  - Crea un nuevo Pokémon en el sistema.
  - Los tipos deben pertenecer al enum `PokemonType`.
- `pokemons`:
  - Devuelve una lista paginada de Pokémon.
  - Los parámetros `page` y `size` son opcionales.
  - Si no se indican, se devolverá por defecto la primera página con 10 pokemons o los existentes en caso de ser menos.
- `pokemon`:
  - Devuelve un Pokémon concreto por su ID.
  - Si no existe, devuelve `null`.

3. Captura y gestión de Pokémon

- `catchPokemon`: (Con autenticación)
  - Permite al entrenador autenticado capturar un Pokémon existente.
  - Se genera un nuevo `OwnedPokemon` con estadísticas (attack, defense, speed, special, level). Estas cuatro variables serán, cada una, un número aleatorio del 1 al 100.
  - OwnedPokemon guardará en la variable pokemon solo el id del pokemon general con el que está relacionado.
  - Dicho pokemon se debe añadir al equipo pokemon del entrenador, un entrenador solo puede tener 6 pokemons en su equipo. Si intenta capturar mas devolverá error.
- `freePokemon`: (Con atenticación)
  - Libera un Pokémon capturado por el entrenador.
  - Solo puede liberar Pokémon que le pertenezcan.
  - Devuelve el entrenador actualizado sin dicho pokemon en su equipo. Se deberán de borrar los datos de dicho pokemon en su completitud además de eliminarlo del equipo de su entrenador.

4. Consulta del entrenador

- `me`: (Con autenticación)
  - Devuelve la información del entrenador autenticado.
  - Incluye todos los Pokémon capturados.
  - Si no hay usuario autenticado, devuelve `null`.
  - Tener en cuenta que en base de datos el equipo solo será un array de id's relacionados con objetos `OwnedPokemon.`

UNIVERSIDAD NEBRIJA

Requisitos técnicos

- Uso de **Node.js** y **Apollo Server**
- Implementación correcta de **resolvers**
- Manejo de errores GraphQL
- Implementación y uso de encadenados según lo pide el esquema (indicado con comentarios para mayor claridad)
- Control de acceso mediante autenticación
- Persistencia de datos (mongodb). Se recomiendo crear tres colecciones, entrenadores, pokemons y ownedPokemons
- Código limpio, modular y correctamente estructurado
- En la entrega se deberá mandar única y exclusivamente el enlace al repositorio de github y las variables de entorno necesarias para la ejecución del código.

Criterios de evaluación

- Corrección funcional de la API
- Uso adecuado de GraphQL
- Seguridad básica (autenticación y autorización)
- Claridad y organización del código
- Correcta implementación del esquema proporcionado
- El examen se corregirá de forma automatizada mediante el código de este repositorio https://github.com/YBlas/ExamenFinalAutomatizado Podréis consultarlo durante el examen para confirmar el estado de vuestro examen. También habrá revisión manual por mi parte, por eso la herramienta evalúa sobre 9.
- El hecho de subir node_modules o el archivo .env al repositorio final entregado tendrá de consecuencia un 0 automático como nota final.