Seminario de Lenguajes Opción: PHP, React y API Rest 2025

API REST / Slim

Slim es un micro framework de PHP que se utiliza para crear aplicaciones web y APIs REST.
Las API REST siguen el modelo CRUD, que significa Create (Crear), Read (Leer), Update (Actualizar) y Delete (Eliminar)
Los recursos de una API REST se identifican mediante endpoints o URI (Uniform Resource Identifier). Cada recurso tiene una dirección única a la que se puede acceder para realizar operaciones sobre él.
Generalmente, se utiliza JSON (JavaScript Object Notation) como formato para el intercambio de datos entre el cliente y el servidor.

ACLARACIONES
Se deben tener las siguientes consideraciones:

- Todos los endpoints deberán tener los métodos y nombres que se detallan en cada ítem.
- Las eliminaciones sólo se podrán ejecutar correctamente en los casos en que el dato no esté siendo utilizado en otra tabla, caso contrario se debe informar el error.
- Después del AppFactory create de slim agregar $app->addBodyParsingMiddleware(); para facilitar la extracción de datos en formato json del body de cada service con $data = $request->getParsedBody();

IMPORTANTE
En todos los casos, los endpoints deberán devolver el estado de la petición realizada, indicando en el header el status code correspondiente:

- 200 OK: Indica que la solicitud se procesó correctamente y se devolvió una respuesta exitosa.
- 400 Bad Request: Se utiliza cuando la solicitud enviada por el cliente es incorrecta o no se puede procesar debido a parámetros faltantes, valores inválidos o problemas de formato.
- 401 Unauthorized: indica que un usuario no puede realizar la acción en cuestión porque no está autorizado.
- 404 Not Found: Indica que el recurso solicitado no se pudo encontrar en el servidor. No existe el endpoint o el recurso (por ejemplo, se buscaron las estadísticas de un jugador con un id que no existe).
- 409 Conflict: Indica que el registro no pudo ser eliminado.

En los casos en que se produce un error, además, deberán devolver un mensaje indicando por qué no se puedo realizar la acción. Es necesario que el código de respuesta se ponga en el servicio para que dicho código sea enviado en un header http sino no será interpretado correctamente cuando se construya el front-end. Por ejemplo:
- No se puede usar una carta que no pertenezca al mazo del usuario logueado.

PROYECTO

Se desea desarrollar una aplicación web que permita administrar y jugar online el siguiente juego de cartas. La base de datos se encuentra precargada con una lista de cartas monstruo, estas tienen un atributo que está presente en la tabla “atributo”. Cada usuario puede registrarse en la web, seleccionar cartas a partir de una lista y crear su mazo indicando las cartas que quiere agregar al mismo.

El juego consiste en crear un mazo de 5 cartas para jugar contra el servidor, quien también tendrá su mazo preestablecido. Un jugador puede crear hasta 3 mazos y cuando desee jugar deberá escoger solo uno de estos.

Una vez elegido el mazo, se iniciará la partida. Cada jugador inicia con sus 5 cartas en mano; en cada ronda debe elegir un monstruo para luchar, al mismo tiempo el servidor también hará su jugada. Algunos atributos de monstruo tienen ventaja sobre otros (provista en la tabla “gana_a”), por ejemplo, el atributo “fuego” es fuerte contra “planta”. Si una carta “fuego” se enfrenta a una carta “planta”, al monstruo con atributo “fuego” se le adiciona 30% de fuerza de ataque sobre su ataque original.

El jugador que baje el monstruo con más ataque, gana la ronda. Una vez terminada la ronda, las cartas usadas se descartan. Quien más victorias tenga luego de las 5 jugadas gana el juego.

Mano Servidor	Mano Jugador
Arcanine (ataque: 110 atributo: fuego) Charizard (ataque: 84 atributo: fuego) Golem (ataque: 120 atributo: piedra) Lapras (ataque: 85 atributo: agua) Onix (ataque: 45 atributo: piedra)	Victreebel (ataque: 105; atributo: planta) Fearow (ataque: 90; atributo: volador) Aerodactyl (ataque: 105; atributo: volador) Arcanine (ataque: 110; atributo: fuego) Tauros (ataque: 100; atributo: normal)

Ejemplo:

Jugada 1:

El servidor juega a Lapras (85, agua) y el jugador a Fearow (90, volador). Fearow tiene más puntos de ataque que Lapras y ningún atributo tiene ventaja sobre el otro. Resultado: Jugador gana.

Jugada 2:

El servidor juega a Charizard (84, fuego) y el jugador a Victreebel (105, planta). Victreebel tiene más puntos de ataque pero el atributo fuego tiene ventaja sobre el atributo planta. Charizard gana +30% = 109,2. Resultado: Servidor gana.

Cómo ayuda, el jugador puede ver los atributos de los monstruos que el contrincante tenga en mano; estos se actualizarán a medida que los monstruos se vayan descartando.
Nota: El usuario servidor/administrador tendrá id = 1; este no puede ser modificado, tampoco la composición de su mazo.

Para esta primera etapa, se pide

a) Desarrollar la función: **jugadaServidor(): int**. La función debe devolver el id de la carta jugada por el servidor. Para que sea considerada válida, la carta debe cumplir las siguientes condiciones:
    - que pertenezca al mazo del servidor, y
    - que no haya sido utilizada previamente en la partida.

   Además debe actualizar el estado de dicha carta en la tabla “mazo_carta” a “descartado”.

b) proveer los siguientes endpoints para comunicarse con el servidor:

Usuarios

- POST /login: Verificar credenciales y retornar token.
    - Cuerpo de la solicitud: JSON con los campos nombre, usuario y clave.
    - Respuesta:
        - Éxito: Código 200 OK, JSON con un token de acceso (token). Se genera un nuevo token, se guarda en la base de datos y se pone vencimiento_token a 1 hora en el futuro.
        - Fallo: Código de estado 401 Unauthorized, con un mensaje de error.

  Luego, cuando se requiera realizar una acción autorizada (como por ejemplo, crear un mazo) se deberá invocar al endpoint con el token recibido.
- POST /registro: Agrega un nuevo usuario con su nombre de usuario y clave.
    - El nombre de usuario debe tener por lo menos 6 caracteres y no más de 20.
    - Únicamente alfanuméricos.
    - Verificar que el nombre de usuario no esté en uso.
    - La clave debe tener por lo menos 8 caracteres y contener caracteres mayúsculas, minúsculas, números y caracteres especiales.
- PUT /usuarios/{usuario}: Editar al usuario logueado (solo nombre y password).
- GET /usuarios/{usuario}: Obtener información del usuario logueado.
    - En los últimos 2 casos (donde se recibe el id) se debe validar que el usuario se haya logueado.

Juego (las operaciones deben validar que el mazo pertenece al usuario logueado)

- POST /partidas

    - Recibe un id de mazo (verificar que pertenezca al usuario logueado)
    - Crea la partida para el usuario logueado, con fecha actual y estado “en_curso”.
    - Actualiza el estado de las cartas de la tabla “mazo_carta” correspondiente al mazo del jugador a estado “en_mano”.
    - Devuelve:
        - Éxito: Código 200 OK como cabecera. En el cuerpo se debe devolver un JSON con id de partida creada y la lista de cartas que usará el usuario logueado.
        - Error: Código de error correspondiente como cabecera. En el cuerpo se debe devolver un mensaje que explique el error.
    - Se debe validar que el usuario se haya logueado.
- POST /jugadas
    - Recibe la carta jugada por el usuario y el id la partida.
    - Verifica que la carta enviada sea válida para jugar (que esté en el mazo con el que el jugador inició la partida y que no haya sido previamente utilizada por el jugador en la partida)
    - Crea un registro en la tabla “jugada”.
    - Debe invocar a la función previamente implementada **jugadaServidor():int**
    - A partir del resultado de la función anterior, analiza cuál es la carta ganadora
    - Actualiza el estado de la carta en la tabla “mazo_carta” a estado “descartado”.
    - Guarda en el registro “jugada” recientemente creado el estado final de la misma: “gano”, “perdio” o “empato”.
    - Si, es la quinta jugada debe cerrar la partida con el estado correspondiente (“finalizada”).
    - Devuelve los siguiente datos:
        - Éxito: Código 200 OK como cabecera. En el cuerpo se debe devolver un JSON con
            - la carta jugada por el servidor.
            - los puntos de fuerza totales de ambas cartas jugadas.
            - Si es la quinta jugada, devolver quien ganó el juego.
        - Error: Código de error correspondiente como cabecera. En el cuerpo se debe devolver un mensaje que explique el error.
    - Se debe validar que el usuario esté logueado

- GET /usuarios/{usuario}/partidas/{partida}/cartas indica los atributos de las cartas que quedan en mano del usuario. Si el usuario es = 1 se supone al jugador servidor. (opcional)
    - Se debe validar que el usuario esté logueado.

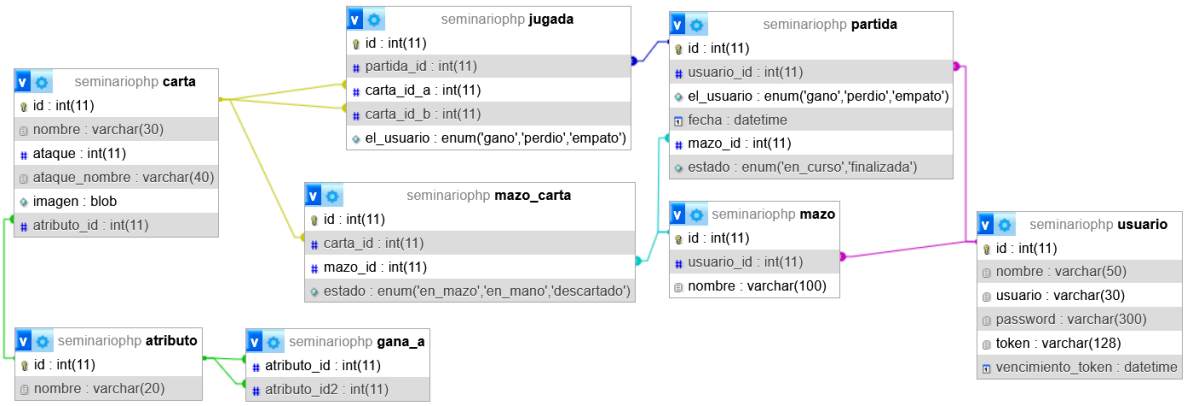
Mazo

- POST /mazos
    - Recibe los ids de cartas que conformarán el mazo y un nombre para identificarlo.
    - Debe validar un máximo de 5 ids de cartas enviadas, deben existir y tener distintos ids.
    - Debe validar que el usuario tenga a lo sumo 3 mazos creados (Max 3 mazos por jugador).
    - La función debe dar de alta un mazo para el usuario y las tuplas correspondientes en la tabla “mazo_carta” (el estado por defecto es “en_mazo”).
    - En caso de ser exitoso el insert, devuelve el id del mazo creado y el nombre. Caso contrario indica error.
    - Validar que el usuario esté logueado.

- DELETE /mazos/{mazo} //Si el mazo ha participado de una partida, no puede borrarse y debe devolver la excepción correspondiente.
    - Validar que el usuario esté logueado.
- GET /usuarios/{usuario}/mazos //devuelve un listado de mazos creados por el usuario logueado
- PUT /mazos/{mazo}
    - nombre
    - Validar que el usuario esté logueado
- GET /cartas?atributo={atributo}&nombre={nombre} // Listar las cartas según los parámetros de búsqueda incluyendo los puntos de ataque.

Estadísticas (no requiere login)

- GET /estadisticas //devuelve cantidad de partidas ganadas, empatadas y perdidas de todos los usuarios.



METODOLOGÍA PARA LA ENTREGA EN IDEAS

- Hacer un archivo zip o rar con todos los archivos del proyecto sin incluir la carpeta vendor. Si se utilizan librerías externas agregar un README indicando para qué las utilizaron y su instalación
- Subir a la sección del grupo asignada (En el repositorio general NO van)
- Fecha de entrega: 06/05