## Primera entrega - API Rest

**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), Retrieve (Recuperar), 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.

**IMPORTANTE**

En todos los casos, los endpoints deberán devolver el estado de la petición realizada:

- **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.
- **409 Conflict:** Indica que el registro no pudo ser eliminado o que la reserva no se pudo hacer porque algún usuario tenía una reserva en el mismo horario en otra cancha.

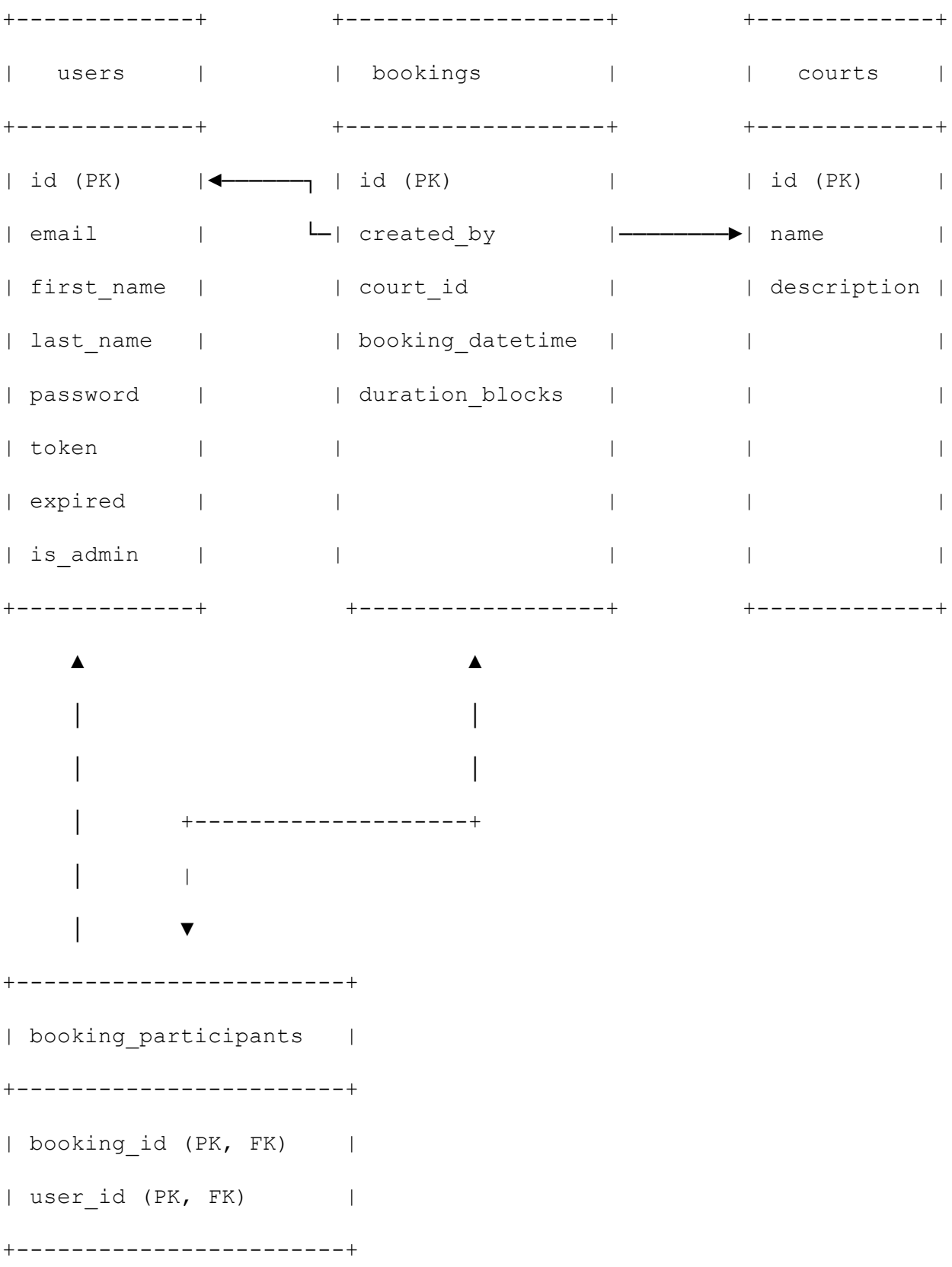
En los casos en que se produce un error, además, deberán devolver un mensaje indicando por qué no se pudo realizar la acción.

**PROYECTO**

Se desea desarrollar una aplicación web que permita a los usuarios alquilar canchas de tenis y encontrar compañeros para jugar (un solo compañero para jugar singles o 3 para jugar dobles).

Un usuario registrado podrá hacer una reserva para una cancha en una fecha y hora específicas (en bloques de media hora que pueden comenzar a las 0 minutos o a las 30 minutos). Las canchas se pueden alquilar desde las 8:00 hasta las 22:00. Todos los tenistas que vayan a jugar en una cancha deben ser usuarios del sistema.

**ERD**


users
id (PK)
email
first_name
last_name
password
token
expired
is_admin
bookings
id (PK)
created_by
court_id
booking_datetime
duration_blocks
courts
id (PK)
name
description
booking_participants
booking_id (PK, FK)
user_id (PK, FK)

**LOGIN**

El mecanismo de autenticación debe funcionar de la siguiente forma:

- **POST /login:**
  - Cuerpo de la solicitud: JSON con los campos `email` y `password`.
  - Respuesta:
    - Éxito: Código 200 OK, JSON con un token de acceso (`token`). Se genera un nuevo token (string aleatorio), se guarda en la base de datos y se pone `expired` a 5 minutos 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, reservar una cancha) se deberá invocar al endpoint con el token recibido (`Authorization: Bearer <token>`) y verificar que no esté vencido. Además, si la operación requiere de un usuario administrador hay que verificar que el usuario sea administrador (is_admin en 1). Además, en cada interacción de un usuario logueado se debe actualizar el campo `expired` 5 minutos en el futuro. Un usuario que no interactúa con el sistema por más de 5 minutos quedará automáticamente deslogueado.

**BACKEND**

- **Login**
  - `POST /login`: Verificar credenciales y en caso de éxito hacer lo antes mencionado.
  - `POST /logout`: Elimina el token de tabla de usuarios y la fecha de expiración.
- **Usuarios**
  - `POST /user`: Crear un nuevo usuario no administrador. Se debe validar que el email tenga el carácter @ y que la clave tenga por los menos 8 caracteres con por lo menos una mayúscula, una minúscula, un número y un carácter especial. Se debe validar que no exista otro usuario con el mismo email.
  - `PATCH /user/{id}`: Editar un usuario existente. Requiere ser administrador o ser el usuario logueado. Se deben enviar **solo** los datos a modificar.
  - `DELETE /user/{id}`: Eliminar un usuario. Requiere ser administrador o ser el usuario logueado y verificar que el usuario no esté en ninguna reserva (en este caso no se podrá borrar). Los usuarios administradores no se pueden borrar.
  - `GET /user/{id}`: Obtener información de un usuario específico. Requiere ser administrador o ser el usuario logueado.
  - `GET /users?search={text}:` Lista todos los usuarios que no son administradores. Además, si el parámetro search es distinto de vacío solo se deberán devolver aquellos que tengan el texto buscado total o parcialmente en el nombre, apellido o email.

- **Canchas**
  - `POST /court`: Crear una nueva cancha. Requiere ser administrador. No se debe permitir crear una cancha con el mismo nombre de una existente.
  - `PUT /court/{id}`: Editar una existente. Requiere ser administrador.
  - `DELETE /court/{id}`: Eliminar una cancha. Requiere ser administrador y verificar que la cancha no esté en ninguna reserva.
  - `GET /court/{id}`: Obtener información de una cancha específica.
- **Reservas**
  - `POST /booking`: Crear una nueva reserva. Cualquier usuario logueado lo puede hacer eligiendo un compañero (single) o a tres (dobles). Se debe validar que el bloque horario que quiere reservar el usuario esté libre. Que no sea de más de 6 bloques de media hora (máximo 3 horas) y que el bloque no exceda el límite de 22:00hs. También se debe validar que el usuario que está haciendo la reserva o alguno de los compañeros no tenga una reserva en otra de las canchas solapadas con el horario que está intentando reservar. *Por ejemplo, si el usuario A quiere hacer una reserva en la cancha 1 hoy de 18:00 a 20:00 con el compañero B y el usuario B ya está en una reserva en la cancha 2 que va de 17:30 a 18:30 no lo podrá hacer porque hay un bloque de media hora (18 a 18:30) en colisión. En este caso el endpoint devolverá 409 y el/los mensaje/s correspondiente/s descriptivos del error.*
  - `DELETE /booking/{id}`: Eliminar una reserva. Solo lo puede hacer el que la hizo o un administrador.
  - `GET /booking?date={date}:` Devuelve todas las reservas del día pasado como parámetro ordenadas por nombre de la cancha y luego por booking_datetime de manera ascendente. No requiere de un usuario logueado.
- **Participantes**
  - `PUT /booking_participant/{id}`: Modifica los participantes de una reserva. Requiere validar que el usuario logueado sea el que la haya creado y que los nuevos participantes no entren en colisión con reservas en las otras canchas.

**NOTAS:**

- El modelo de la base de datos es entregado por la cátedra y no debe modificarse.
- El creador de la reserva también tiene un registro booking_participants
- En booking_participants para una determinada reserva tiene que haber 2 o 4 registros de todos los que van a jugar single o dobles.
- Cada reserva tiene un solo registro en bookings, por ejemplo:

```
{
  "id": 12,
  "created_by": 3,
  "court_id": 5,
  "booking_datetime": "2025-08-06 18:30:00",
  "duration_blocks": 2
}
```


que quiere decir que la reserva va de 18:30 a 19:30 (dos bloques de media hora).

## METODOLOGÍA PARA LA ENTREGA EN IDEAS

- Hacer un archivo .zip o .rar con todos los archivos del proyecto.
- Subir al repositorio del grupo, no al repositorio general.

## SCRIPT PARA CREAR LA BASE DE DATOS

```
-- Creamos el esquema si no existe
CREATE SCHEMA IF NOT EXISTS `seminariophp`;
USE `seminariophp`;

-- Tabla de usuarios
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  first_name VARCHAR(100) NOT NULL,
  last_name VARCHAR(100) NOT NULL,
  password VARCHAR(255) NOT NULL,
  token VARCHAR(255) UNIQUE,
  expired DATETIME,
  is_admin BOOLEAN NOT NULL DEFAULT FALSE
);

-- Tabla de canchas
CREATE TABLE courts (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  description TEXT
);

-- Tabla de reservas
CREATE TABLE bookings (
  id INT AUTO_INCREMENT PRIMARY KEY,
  created_by INT NOT NULL,
  court_id INT NOT NULL,
  booking_datetime DATETIME NOT NULL,
  duration_blocks INT NOT NULL,
  FOREIGN KEY (created_by) REFERENCES users(id),
  FOREIGN KEY (court_id) REFERENCES courts(id)
);

-- Tabla de participantes en cada reserva
CREATE TABLE booking_participants (
  id INT AUTO_INCREMENT PRIMARY KEY,
  booking_id INT NOT NULL,
  user_id INT NOT NULL,
  UNIQUE KEY unique_booking_user (booking_id, user_id),
  FOREIGN KEY (booking_id) REFERENCES bookings(id),
  FOREIGN KEY (user_id) REFERENCES users(id)
);
```