REC. SEGUNDO PARCIAL – PROGRAMACIÓN III – 2 cuat. 2023

Aclaración:
Las partes se corregirán de manera secuencial (ascendentemente). Si están bien todos los puntos de una parte, habilita la corrección de la parte posterior.
Se debe crear un archivo por cada entidad de PHP. Todas las entidades deben estar dentro de un namespace (apellido\nombre del alumno).
Todos los métodos deben estar declarados dentro de clases.
PDO es requerido para interactuar con la base de datos.
Se deben respetar los nombres de los archivos, de las clases, de los métodos y de los parámetros de las peticiones.
Utilizar Slim framework 4 para la creación de la Api Rest.
El virtual host definirlo cómo → http://slim4_parciales.
Respetar la estructura de directorios (index.php → ./public; fotos → ./src/fotos; clases en ./src/poo;).
NO agregar lógica dentro de los callbacks de la API, referenciar métodos de las clases correspondientes.
Habilitar .htaccess dónde corresponda.

Parte 1 (hasta un 6)

Crear un ***APi Rest*** para la administración de **usuarios** y **perfiles**. La interacción será con la clase ***Perfil***, la clase ***Usuario*** y la base de datos ***administracion_bd*** (*perfiles* - *usuarios*).
Crear los siguientes verbos:

A nivel de ruta (/usuario):
(POST) Alta de usuarios. Se agregará un nuevo registro en la tabla usuarios *.
Se envía un JSON → **usuario** (correo, clave, nombre, apellido y id_perfil **) y **foto**.
La foto se guardará en ./src/fotos, con el siguiente formato: ***id_apellido.extension***.
Ejemplo: ./src/fotos/24_perez.jpg
* ID auto-incremental. ** 1,- propietario, 2,- encargado, 3.- empleado.
Retorna un JSON (éxito: true/false; mensaje: string; status: 200/418)

A nivel de aplicación:
(GET) Listado de usuarios. Mostrará el listado completo de los usuarios (array JSON).
Retorna un JSON (éxito: true/false; mensaje: string; dato: stringJSON; status: 200/424)

A nivel de aplicación:
(POST) Alta de perfiles. Se agregará un nuevo registro en la tabla perfiles *.
Se envía un JSON → **perfil** (descripción y estado **).
* ID auto-incremental. ** 1 → Activo y 0 → Inactivo.
Retorna un JSON (éxito: true/false; mensaje: string; status: 200/418)

A nivel de ruta (/perfil):
(GET) Listado de perfiles. Mostrará el listado completo de los perfiles (array JSON).
Retorna un JSON (éxito: true/false; mensaje: string; dato: stringJSON; status: 200/424)

A nivel de ruta (/login):
(POST) Se envía un JSON → **user** (correo y clave) y retorna un JSON (éxito: true/false; jwt: JWT (con todos los datos del usuario, a excepción de la clave) / null; status: 200/403)

(GET) Se envía el JWT → **token** (en el **Bearer**) y se verifica. En caso exitoso, retorna un JSON con éxito (true/false) y status (200/403).

NOTA: Todos los verbos invocarán métodos de la clase Perfil o Usuario para realizar las acciones.

En la creación del JWT, establecer un valor de expiración de 45 **segundos**.

Crear el grupo **/perfiles** con los siguientes verbos:

(DELETE) Borrado de perfiles por ID.
Recibe el ID del perfil a ser borrado (**id_perfil**, cómo parámetro JSON) más el JWT → **token** (en el **Bearer**).
Retorna un JSON (éxito: true/false; mensaje: string; status: 200/418)

(PUT) Modificación de perfiles por ID.
Recibe el JSON del perfil a ser modificado → **perfil** (descripcion y estado), el ID → **id_perfil** (id del perfil a ser modificado) y el JWT → **token** (en el **Bearer**).
Retorna un JSON (éxito: true/false; mensaje: string; status: 200/418)

Parte 2 (hasta un 8)

Crear el grupo **/usuarios** con los siguientes verbos:

(DELETE) Borrado de usuarios por ID.
Recibe el ID del usuario a ser borrado (**id_usuario**, cómo parámetro JSON) más el JWT → **token** (en el **Bearer**).
Retorna un JSON (éxito: true/false; mensaje: string; status: 200/418)

(POST) Modificar los usuarios por ID.
Recibe el JSON del usuario a ser modificado → **usuario** (id, correo, clave, nombre, apellido y id_perfil), la **foto** y el JWT → **token** (en el **Bearer**).
Retorna un JSON (éxito: true/false; mensaje: string; status: 200/418)

Crear un Middleware (en la clase **MW**) para que:

(**método de instancia**) verifique que el **token** sea válido.
Recibe el JWT → **token** (en el **Bearer**) a ser verificado.
Retorna un JSON con el mensaje de error correspondiente (y status 403), en caso de no ser válido.
Caso contrario, pasar al siguiente callable.

Aplicar el Middleware a los grupos **/perfiles** y **/usuarios.**