FINAL de PROGRAMACIÓN III 2024

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_finales.
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 01 (hasta un 6)

Crear un API Rest para la gestión de un kiosco, que interactúe con la clase ***Articulo***, la clase ***Usuario*** y la base de datos ***kiosco_bd*** (articulos - usuarios).
Crear los siguientes verbos:

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

A nivel de aplicación:
(POST) Alta de artículos. Se agregará un nuevo registro en la tabla *articulos*.
Se envía un JSON → ***articulo_json*** (codigo_barra, nombre y precio) y **foto**.
La foto se guardará en ***./src/fotos***, con el siguiente formato: ***codigo_barra_nombre.extension***.
Ejemplo: ***112233_alfajor.jpg***.
Retorna un JSON (éxito: true/false; mensaje: string; status: 200/418)

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

(POST) Modificar artículo.
Recibe el JSON del artículo a ser modificado → ***articulo_json*** (codigo_barra, nombre y precio) y **foto**.
Retorna un JSON (éxito: true/false; mensaje: string; status: 200/418)

(DELETE) Borrar artículo.
Recibe el código de barra del artículo a ser borrado (***codigo_barra***, cómo parámetro de ruta)
Retorna un JSON (éxito: true/false; mensaje: string; status: 200/418)

A nivel de ruta (**/login**) crear los siguientes verbos:
(POST) Se envía un JSON → ***user*** (legajo y clave), verifica en la tabla ***usuarios*** si existe o no.
Retorna un JSON (éxito: true/false; jwt: JWT (*) / null; status: 200/403).

(*) el payload del JWT tendrá la siguiente estructura:

- usuario: con todos los datos del usuario, a excepción de la clave.
- alumno: colocar su nombre y apellido
- dni_alumno: indicar su número de DNI
- fecha_final: indicar la fecha del final (aaaa/mm/dd)

El token creado deberá ser firmado con el ***apellido.nombre*** del alumno y debe expirar en **120** segundos.

(GET) Se envía el JWT → (en el **Bearer**) y se verifica. Retorna un JSON (éxito: true/false; status: 200/403).

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

Parte 02 (hasta un 8)

Crear los siguientes ***Middlewares*** (en la clase MW) para que:
1.- (método de clase) Si alguno de los campos ***legajo*** o ***clave*** están vacíos (o los dos) retorne un JSON con el mensaje de error correspondiente (y status 409).
Caso contrario, pasar al siguiente callable.
2.- (método de instancia) Verifique que el token sea válido. Recibe el JWT → (en el **Bearer**) a ser verificado. En caso de no ser válido, retorna un JSON con el mensaje de error correspondiente (y status 403).
Caso contrario, pasar al siguiente callable.
3.- (método de clase) Verifique que el ***código de barra*** NO exista en la base de datos. Si EXISTE, retornar un JSON con el mensaje de error correspondiente (y status 403).
Caso contrario, acceder al siguiente callable.
Crear, a nivel de grupo (**/mw**), las siguientes rutas y verbos:
A nivel de ruta (**/login_mw**):
A partir del verbo POST de **/login**, aplicarle el middleware 1.

A nivel de ruta (**/crud**):
A partir del verbo POST (a nivel de aplicación), aplicar los middlewares 2 y 3.
A nivel de ruta (**/crud_mw**)
A partir del verbo POST de **/articulo**, aplicar el middleware 2.
A partir del verbo DELETE de **/articulo**, aplicar el middleware 2.

Parte 03

A nivel de ruta **(/pdf)**:
(GET) Listado en formato .pdf.
Se envía el JWT → (en el **Bearer**) y mostrará:

*-Encabezado (apellido y nombre del alumno a la izquierda y número de página a la derecha)
*-Cuerpo (Título del listado, listado completo de los artículos con su respectiva foto)
*-Pie de página (fecha y hora actual, centrada).

NOTA: El archivo .pdf contendrá clave, si el rol es empleado, será el apellido del usuario logueado, si el rol es supervisor, será la clave del usuario logueado y si el rol es administrador, será el legajo del usuario logueado.
Agregar el middleware 2 (de la parte anterior) para que verifique el JWT.