API REST PHP Y MARIADB

CON TOKEN DE SEGURIDAD JWT

Ya en el ejemplo: “API_REST_PHP” colgado también acá en Github, se puede apreciar bien, un clásico uso del Api Rest en PHP con MariaDB, sin embargo, ese ejemplo es muy simple, puesto que allí se manejó una estructura en BD muy sencilla, con solo tres columnas, donde simulamos con los nombres de jugadores de la liga de baloncesto NBA.

Ahora vamos a hacer lo mismo, conservando las capas, arquitectura y esquema de funcionamiento, pero añadiendo algo de complejidad, se va a trabajar con una estructura con más columnas, para manipular un JSON más robusto, y le incluiremos seguridad al REST con autenticación y autorización, mediante un token basado en JWT.

La seguridad de las API es imprescindible, pues es necesario protegerlas de los ataques. Al igual que las aplicaciones, las redes y los servidores pueden ser objeto de ataques, y las API desde luego no están exentas de ser víctimas de diferentes amenazas.

Como es bien sabido, son múltiples los ataques a los que están expuestas las aplicaciones web y los servidores que las alojan, existen ataques de XSS Cross-Site Scripting, Cross-Site Request Forgeries, Session Hijacking y por supuesto SQL Injection.

Con la autentificación se busca verificar la identidad del usuario o proceso que ejecuta el API, mientras que la autorización valida los privilegios de acceso del usuario a los recursos del API.

Hay muchos métodos de seguridad y muchas formas de autentificarse, que pueden depender desde el tipo de dispositivo, el tipo de uso y la confidencialidad de la información, entre otros. No hay una sola manera de asegurar un API.

Los 4 métodos principales de autentificación API REST son:

1. Autentificación básica
2. Autentificación basada en token
3. Autentificación basada en clave API
4. OAuth 2.0 (Autorización abierta)

Para este ejemplo se tendrá en cuenta la autenticación basada en token, en este método, el usuario se identifica al igual que con la autenticación básica, con sus credenciales, nombre de usuario y contraseña. Pero en este caso, con la primera petición de autenticación, el servidor generará un token basado en esas credenciales.

El servidor guarda en base de datos este registro y lo devuelve al usuario para que a partir de ese momento no envíe más credenciales de inicio de sesión en cada petición HTTP. En lugar de las credenciales, simplemente se debe enviar el token codificado en cada petición HTTP.

Por norma general, los tokens están codificados con la fecha y la hora para que, en caso de que alguien intercepte el token con un ataque MITM, no pueda utilizarlo pasado un tiempo establecido, puesto que el token se puede configurar para que caduque después de un tiempo definido, por lo que los usuarios deberán iniciar sesión de nuevo.

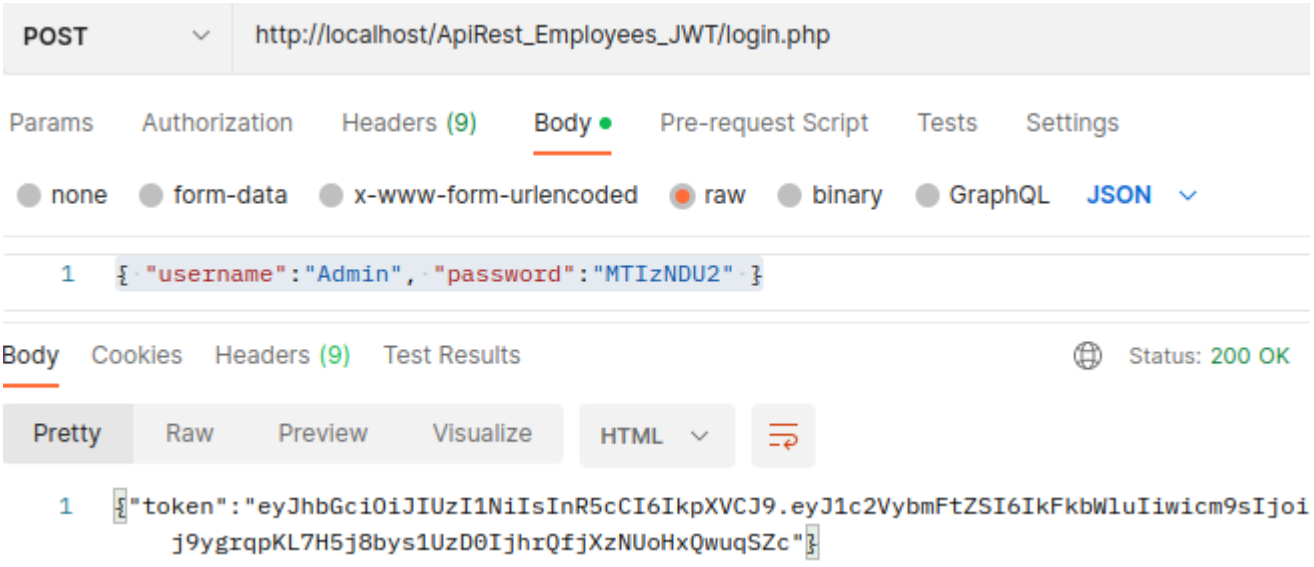
Normalmente se emplean frameworks, y bibliotecas de terceros que ayudan mucho en la implementación de seguridad, sin embargo yo no usaré ninguna biblioteca, aquí haremos uso de un token siguiendo las directrices de JWT (JSON Web Token), con sus habituales header, payload y signature, para autorizar a los usuarios y permitirles continuar con sus trabajos, una vez que hayan iniciado sesión con sus credenciales habituales (nombre de usuario y clave). Los usuarios continuan con su trabajo hasta que el token caduque. Esta adaptación se hizo con base en tutoriales del programador Soumitra, autor de la página roytuts.com

Para implementar esa autentificación y autorización en este ejemplo, creamos una nueva estructura llamada: tbl_security, donde hay una columna llamada: profile, que va a tener dos posibles valores: Admin y Aux.

Solo quien tiene el perfil de Admin puede ejecutar la opción DELETE del Api, se procederá de la siguiente manera:

Cuando el usuario ingresa, en la BD hay un procedimiento almacenado llamado: spr_R_ObtenerAcceso, que valida las credenciales de quien intenta ingresar, y devuelve el username y el perfil de acceso, dicho perfil se almacena en el payload, y cuando intenta efectuar un delete, pregunta por ese valor si es de tipo: Admin.

Esto reflejado en este ejemplo, es que un usuario ingresa un JSON con sus credenciales así:



Eso sí jamás enviar claves y passwords en url mediante protocolo HTTP, **solo hacerlo mediante HTTPS**, y por lo menos codificado en base64; pero aún así a mi no me gusta hacerlo, porque la realidad es que enviar claves y datos sensibles mediante una url, no es del todo sano, así estuviese codificado en base64, esto no lo blinda puesto que la codificación base64 solo enmascara, no cifra y es fácilmente reversible, si bien HTTPS brinda una mayor seguridad, no le hace invulnerable a un ataque.

Yo soy partidario es de crear una aplicación web bien rígida, solo para login y contraseña, y una vez autenticado se genera el token, que a la par se almacena en BD, y se establecen mecanismos de seguridad tanto en cliente como en servidor, para evitar enviar claves en url y/o el JSON propiamente; me gusta manejar claves numéricas a fin de evitar SQL injection, pues este campo se valida para que sea estrictamente numérico, y en BD se manejan otros controles que le den mayor seguridad, como una frase y/o imagen solo conocida porque quien ingresa.

En BD el procedimiento almacenado spr_R_ObtenerAcceso ejecuta este query:

```
SELECT username, profile
FROM tbl_security
WHERE username =  p_username
AND cast(aes_decrypt(password, 'ApiRest_PHP_JWT') as char) = p_password Limit 1;
```

Como las credenciales son válidas, el procedimiento almacenado devuelve el profile, y PHP genera el token, que en el proyecto en PHP se toma en el archivo: login.php y de inmediato ordena la generación del token con la instrucción:

$jwt = generate_jwt($headers, $payload); así:

```
if ($user->getProfile()!=null)
{
    $username = $user->getUsername();
    $perfil = $user->getProfile();
    $headers = array('alg'=>'HS256','typ'=>'JWT');
    $payload = array('username'=>$username, 'rol'=>$perfil, 'exp'=>(time() + 120));
    $jwt = generate_jwt($headers, $payload);
    echo json_encode(array('token' => $jwt));
}
```

El payload almacena el perfil que devolvió el SP, y cuando el usuario intenta acceder a la función eliminarEmpleado en TokenForceAPI.php, se valida si cuenta con autorización para ello.

Acá lo que se hace es que se verifica si es válido el token y **además** si el usuario cuenta con perfil de **Admin.**

```
function eliminarEmpleado(){
    // **==**==**==**==**==**==**==**==**==**==**==**==
    header("Access-Control-Allow-Origin: *");
    header("Access-Control-Allow-Methods: DELETE");

    $bearer_token = get_bearer_token();
    $is_jwt_valid = is_jwt_valid($bearer_token);
    $perfil = verificarPerfil($bearer_token);
    if (($is_jwt_valid) && ($perfil == 'Admin'))
    {
        $longitud = strlen(DOMINIO);
        $host= $_SERVER["HTTP_HOST"];
```

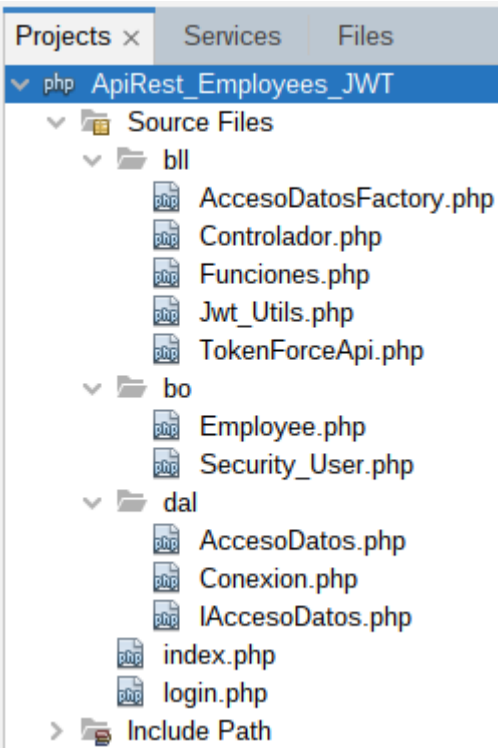
La distribución del ejemplo es exactamente igual al ejemplo: API_REST_PHP, acá solo hay 4 clases adicionales que son:

Jwt_Utils.php: Ubicada dentro del paquete bll o lógica de negocio, que es la clase encargada de generar el Token.

login.php: Es la clase encargada de recibir y validar las credenciales del usuario.

Employee.php: Es la clase que representa la nueva estructura en BD, que fue tomada de los ejemplos del schema HR de Oracle, llamada: employees.

Security_User.php: Es la otra clase que representa la estructura de seguridad en BD.



Es importante aclarar que hay algo que no va a quedar bien hecho, y es la forma como se manejan los perfiles de acceso acá, pues se está haciendo uso de la estructura de acceso de usuario y clave, y en esa misma se aloja el perfil (profile), en el mundo real, un usuario eventualmente puede tener múltiples roles, o permisos sobre un sistema, lo que desde luego da para una relación M:M (Muchos usuarios vs Muchos roles), y ello obliga a normalizar dicha relación, creando una tercera estructura, donde se relacionaría security_id con el id del rol implicado, cosa que se omitió, pero cada quien que mejore y optimice tal situación.

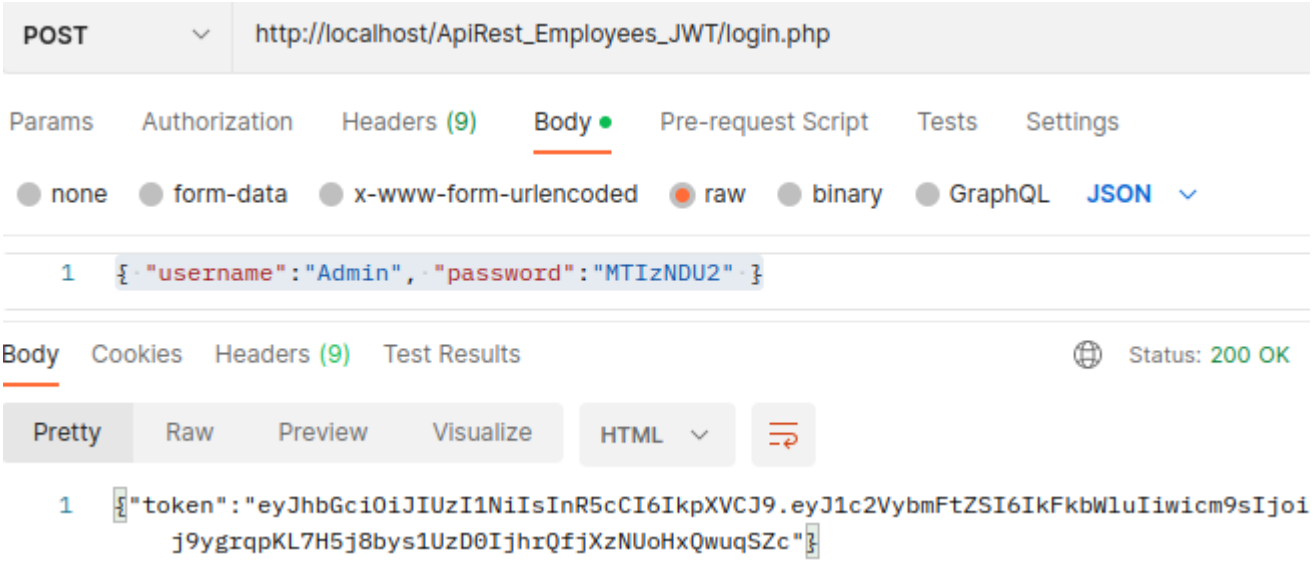
Finalmente hagamos las pruebas con Postman:

Pruebas con Postman

Para todos los casos, requiere autenticarse y debe enviarse JSON mediante POST así:

{ "username":"Admin", "password":"MTIzNDU2" }

http://localhost/ApiRest_Employees_JWT/login.php



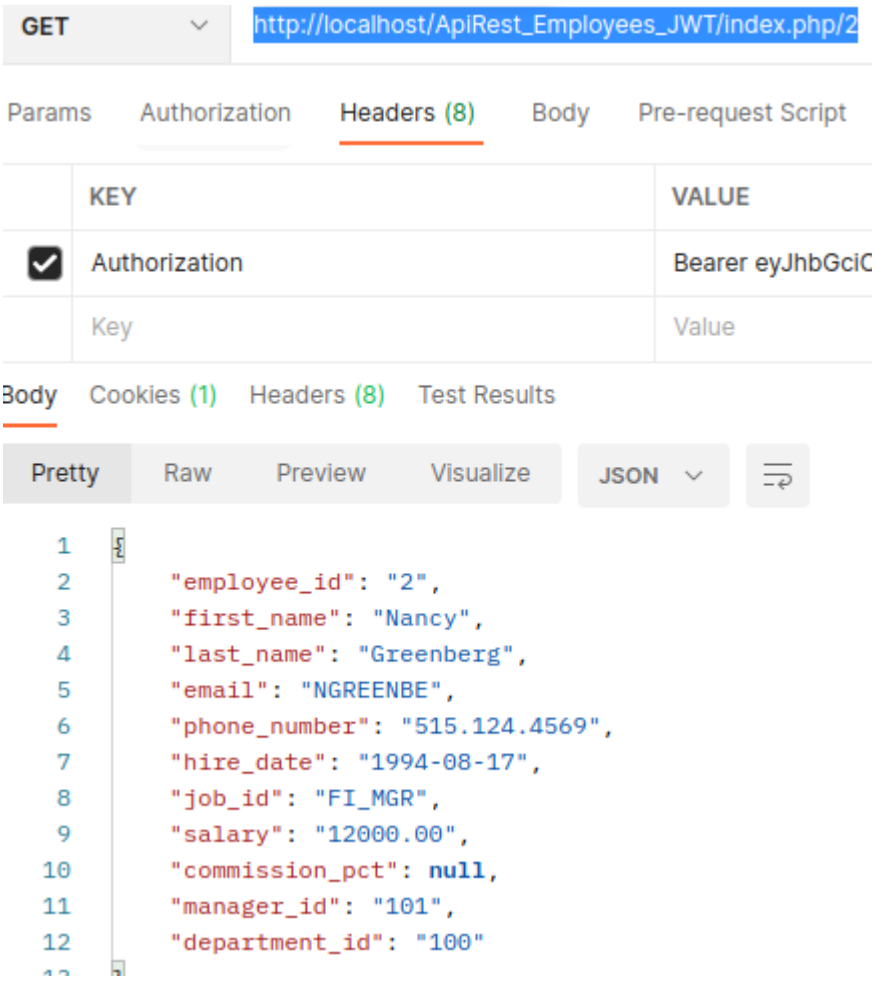
Ahora hacemos un Get de todos los empleados:

http://localhost/ApiRest_Employees_JWT/index.php


```
    {
        "employee_id": "1",
        "first_name": "Steven",
        "last_name": "King",
        "email": "SKING",
        "phone_number": "515.123.4567",
        "hire_date": "1987-06-17",
        "job_id": "AD_PRES",
        "salary": "24000.00",
        "commission_pct": null,
        "manager_id": null,
        "department_id": "90"
    },
    {
        "employee_id": "2",
        "first_name": "Nancy",
        "last_name": "Greenberg",
        "email": "NGREENBE",
        "phone_number": "515.124.4569",
        "hire_date": "1994-08-17",
        "job_id": "FI_MGR",
        "salary": "12000.00",
        "commission_pct": null,
        "manager_id": "101",
        "department_id": "100"
    }
```

Si quisiéramos consultar un empleado en particular, entonces hacemos un GET nuevamente y adelante en la url colocamos el id así:

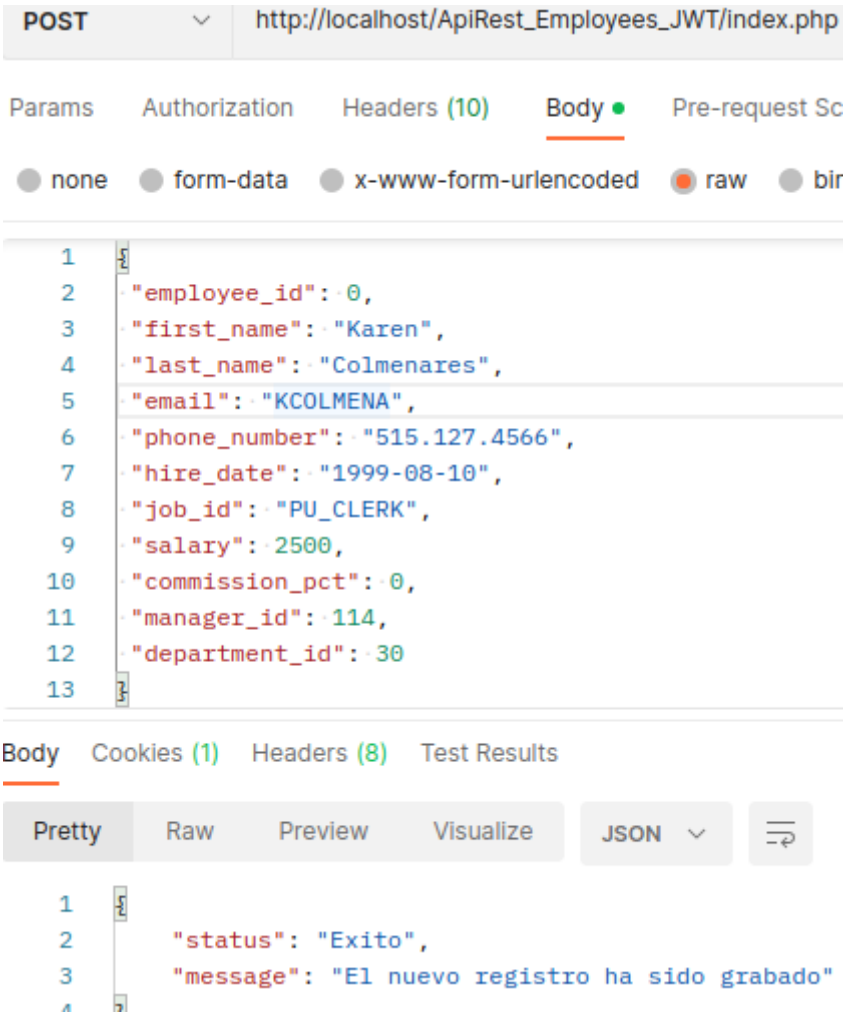
http://localhost/ApiRest_Employees_JWT/index.php/2



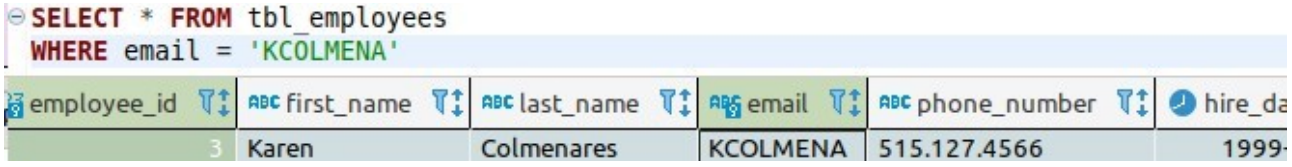
Ahora vamos a crear un empleado, luego de hacer el GET para obtener el token de autorización, ingresamos en el body el siguiente JSON:


```
{
        "employee_id": 0,
        "first_name": "Karen",
        "last_name": "Colmenares",
        "email": "KCOLMENA",
        "phone_number": "515.127.4566",
        "hire_date": "1999-08-10",
        "job_id": "PU_CLERK",
        "salary": 2500,
        "commission_pct": 0,
        "manager_id": 114,
        "department_id": 30
}
```

http://localhost/ApiRest_Employees_JWT/index.php

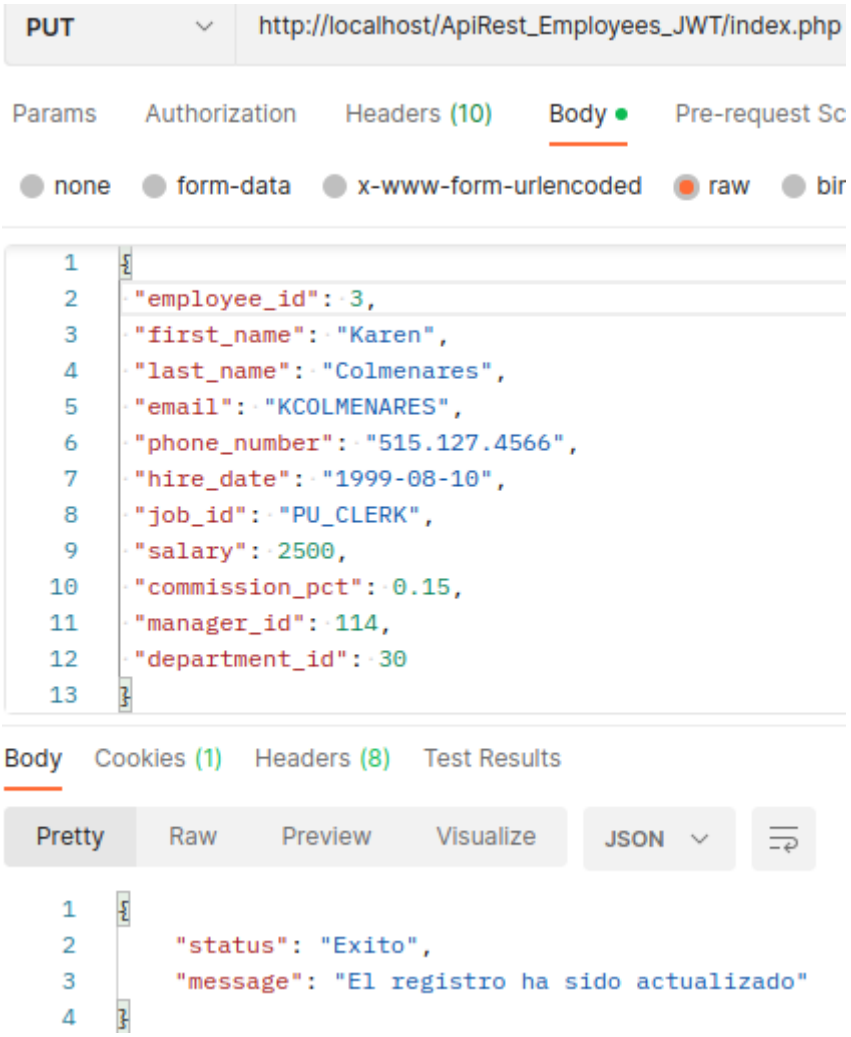


Verificando ese insert en la BD obtenemos lo siguiente:

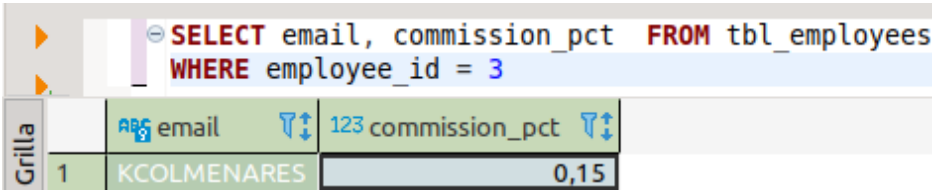


Ahora actualizaremos ese empleado que se acabó de crear, le modificaremos el email de “KCOLMENA” por: “KCOLMENARES", y su comisión que actualmente es 0 o nula, por 0,15 así:

http://localhost/ApiRest_Employees_JWT/index.php

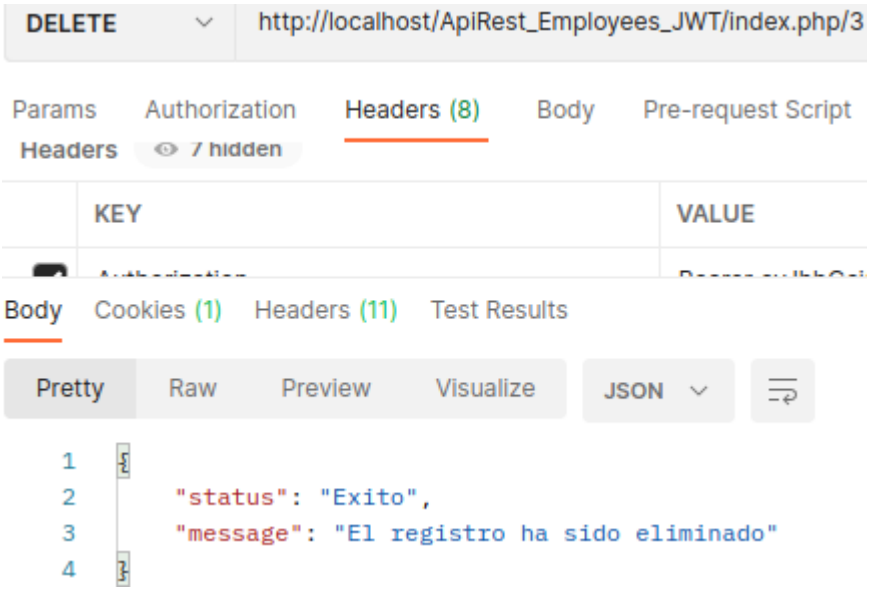


Si verificamos en la BD con un select vemos que la actualización fue exitosa:



Por último eliminemos ese empleado que acabamos de actualizar:

http://localhost/ApiRest_Employees_JWT/index.php/3



De implementar esto en un ambiente de producción, sobra decir que las “secret key” que se elijan, tanto en MariaDB como en PHP, deben ser claves secretas **muy robustas**, y que obviamente solo deben conocer los administradores del servidor y quienes diseñan la BD. Por supuesto todo debe hacerse siempre sobre HTTPS, porque la realidad es que en estos token basados en JWT, las primeras dos partes (header y payload) son strings en base64, cuya base son caracteres ASCII, que sólo enmascaran lo que viaje, pero no es cifrado, lo que no los hace invulnerables, de lógica todo lo que viaje por la red, es susceptible de que alguien pueda ver e interceptar.

Por su parte la firma se basa en el estándar HMAC-SHA256, que hasta ahora ha demostrado ser seguro, pero es importante entender que el propósito de utilizar token basados en JWT, no es para ocultar los datos. El motivo por el que se utilizan es para demostrar que los datos enviados y recibidos fueron creados por una fuente auténtica reconocida. Los datos de la firma permiten verificar al receptor (su aplicación y nuestra API) la autenticidad de la fuente de los datos como tal; sin embargo más se trasnocha alguien, en construir una aplicación supuestamente segura, mientras que hay un atacante en algún rincón del planeta, igualmente trasnochando, pero para con algoritmos de fuerza bruta, tratar de romper esa seguridad…

Finalmente, esta no es ni la mejor, ni la única forma de hacer las cosas, existen muchas implementaciones y adaptaciones con la librería JWT, y por supuesto variedad de frameworks y utilidades, que coadyuvan en la consecución de la autenticación, de quien invoca un API. La seguridad es un asunto muy serio y este ejemplo corto se queda, ya que la seguridad informática es primordial, y cada vez más importante, porque los humanos estamos cada vez más expuestos a la red, sin embargo, todavía son pocos los que se preocupan como deberían por su seguridad y privacidad.

Y como se dice popularmente: “*Ser capaz de superar la seguridad no te convierte en un hacker. Si tu empresa gasta más en café que en seguridad TI, serás hackeado. Es más, mereces ser hackeado.”*