



## **CENTRO DE DESARROLLO SOFTWARE**

### **SEDE SANTA ANA**

#### **Nivel 3 Programador Analista en Lenguaje PHP**

## **Guía de Autenticación con JSON Web Tokens (JWT) en Codeigniter**

**Alumno:** Daniel Alberto Murillo Aguilar

**Instructor:** Noel Morán Galdámez

**Fecha de entrega:** lunes 30 de septiembre 2019.

## Requisitos

1. Framework Codeigniter.
2. Tener las configuraciones y los servicios REST hechos con la librería RESTController para las tablas genero y libro.

## Pasos para crear un sistema de autenticación usando JWT en Codeigniter

### Primer paso:

El primer paso será crear una nueva tabla llamada usuario, en nuestra base de datos “libros\_db\_jwt”.

```
DROP DATABASE IF EXISTS libros_db_jwt;  
CREATE DATABASE libros_db_jwt;  
USE libros_db_jwt;
```

```
CREATE TABLE genero(  
id_genero int primary key not null auto_increment,  
titulo varchar(100)  
);
```

```
CREATE TABLE libro(  
isbn int primary key not null,  
titulo varchar(100) not null,  
autor varchar(100) not null,  
id_genero int not null,  
FOREIGN KEY (id_genero) REFERENCES genero(id_genero)  
);
```

```

CREATE TABLE usuario(
id_usuario int primary key auto_increment not null,
nombre_usuario varchar(64) not null,
contrasenia varchar(16) not null
);

/*Datos de prueba*/
INSERT INTO usuario VALUES (null, 'dani', '12345');
INSERT INTO genero VALUES (null, 'Comedia');
INSERT INTO genero VALUES (null, 'Fantasía');
INSERT INTO genero VALUES (null, 'Tragedia');
INSERT INTO genero VALUES (null, 'Romance');
INSERT INTO libro VALUES(485787489, 'Un librito', 'Dani', 1);

```

## Segundo paso:

Configuraciones básicas del proyecto.

En la carpeta application->config del framework modificaremos varios archivos:

- El archivo autoload.php.

```

$autoload['libraries'] = array('database');
$autoload['helper'] = array('date');
$autoload['config'] = array('jwt');

```

- El archivo config.php.

```

$config['base_url'] = 'http://localhost/php_api_rest';

```

- El archivo database.php.

```

$db['default'] = array(
    'dsn' => 'mysql:host=localhost; dbname=libros_db_jwt; charset=utf8;',
    'hostname' => 'localhost',
    'username' => 'root',
    'password' => '',
    'database' => 'libros_db_jwt',
    'dbdriver' => 'pdo',
    'dbprefix' => '',
    'pconnect' => FALSE,
    'db_debug' => (ENVIRONMENT !== 'production'),

```

```

        'cache_on' => FALSE,
        'cachedir' => '',
        'char_set' => 'utf8',
        'dbcollat' => 'utf8_general_ci',
        'swap_pre' => '',
        'encrypt' => FALSE,
        'compress' => FALSE,
        'stricton' => FALSE,
        'failover' => array(),
        'save_queries' => TRUE
    );

```

- Para hacer uso de PDO en nuestra conexión a la bse de datos y poder manejar las excepciones tendremos que cambiar en el archivo system->database->drivers->pdo->pdo\_driver.php.

```

public $options = array(
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
);

```

- Agregaremos un nuevo archivo llamado jwt.php con el siguiente contenido.

```

<?php
defined('BASEPATH') OR exit('No direct script access allowed');
// Esta es nuestra llave secreta para encriptar el token
$config['jwt_key'] = '123';
// El tiempo de vida del token (en minutos)
$config['token_timeout'] = 1;

```

En la carpeta application->helpers del framework agregaremos dos archivos:

- El archivo authorization\_helper.php tomado de (ParitoshVaidya, 2019): Este contiene los métodos básicos para generar un nuevo token, para validar el tiempo de vida del token, para validar el contenido del token y un método para verificar la solicitudes de la api.

```

<?php
class AUTHORIZATION
{
    public static function validateTimestamp($token)
    {
        $CI = &get_instance();
        $token = self::validateToken($token);
        if ($token != false && (now() - $token->timestamp < ($CI->config->item('token_timeout') * 60))) {

```

```

        return $token;
    }
    return false;
}

public static function validateToken($token)
{
    $CI = &get_instance();
    return JWT::decode($token, $CI->config->item('jwt_key'));
}

public static function generateToken($data)
{
    $CI = &get_instance();
    return JWT::encode($data, $CI->config->item('jwt_key'));
}

public static function verify_request($token)
{
    // Extraemos el token si está seteado en el header
    if (!empty($token)) {
        try {
            // Validamos el token
            // Si la validación es correcta nos devolverá el contenido del token, si no, nos devolverá false
            $data = self::validateToken($token);
            if ($data === false) {
                return REST_Controller::HTTP_FORBIDDEN;
                exit();
            } else {
                return $data = REST_Controller::HTTP_OK;
            }
        } catch (Exception $e) {
            return $data = REST_Controller::HTTP_UNAUTHORIZED;
            exit();
        }
    }
    // sino devolvemos un código de estado 401
    else {
        return $data = REST_Controller::HTTP_UNAUTHORIZED;
        exit();
    }
}
}
}

```

- El archivo jwt\_helper, se ha utilizado sin ninguna modificación del proporcionado por (ParitoshVaidya, 2019).

```

<?php

/**
 * JSON Web Token implementation, based on this spec:
 * http://tools.ietf.org/html/draft-ietf-oauth-json-web-token-06
 *
 * PHP version 5
 *
 * @category Authentication
 * @package Authentication_JWT
 * @author Neuman Vong <neuman@twilio.com>
 * @author Anant Narayanan <anant@php.net>
 * @license http://opensource.org/licenses/BSD-3-Clause 3-clause BSD
 * @link https://github.com/firebase/php-jwt
 */
class JWT
{
    /**
     * Decodes a JWT string into a PHP object.
     *
     * @param string $jwt The JWT
     * @param string|null $key The secret key
     * @param bool $verify Don't skip verification process
     *
     * @return object The JWT's payload as a PHP object
     * @throws UnexpectedValueException Provided JWT was invalid
     * @throws DomainException Algorithm was not provided
     *
     * @uses jsonDecode
     * @uses urlsafeB64Decode
     */
    public static function decode($jwt, $key = null, $verify = true)
    {
        $tkts = explode('.', $jwt);
        if (count($tkts) != 3) {
            //if you don't want to disclose more details
            return false;

            //throw new UnexpectedValueException('Wrong number of segments'
        );
        }
        list($headb64, $bodyb64, $cryptob64) = $tkts;
        if (null === ($header = JWT::jsonDecode(JWT::urlsafeB64Decode($head
b64)))) {
            //if you don't want to disclose more details
            return false;

```

```

        //throw new UnexpectedValueException('Invalid segment encoding'
    );
    }
    if (null === $payload = JWT::jsonDecode(JWT::urlsafeB64Decode($body
b64))) {
        //if you don't want to disclose more details
        return false;

        //throw new UnexpectedValueException('Invalid segment encoding'
    );
    }
    $sig = JWT::urlsafeB64Decode($cryptob64);
    if ($verify) {
        if (empty($header->alg)) {
            //if you don't want to disclose more details
            return false;

            //throw new DomainException('Empty algorithm');
        }
        if ($sig != JWT::sign("$headb64.$bodyb64", $key, $header->alg)) {
            throw new UnexpectedValueException('Signature verification
failed');
        }
    }
    return $payload;
}

/**
 * Converts and signs a PHP object or array into a JWT string.
 *
 * @param object|array $payload PHP object or array
 * @param string      $key      The secret key
 * @param string      $algo     The signing algorithm. Supported
 *                               algorithms are 'HS256', 'HS384' and 'HS
512'
 *
 * @return string      A signed JWT
 * @uses jsonEncode
 * @uses urlsafeB64Encode
 */
public static function encode($payload, $key, $algo = 'HS256')
{
    $header = array('typ' => 'JWT', 'alg' => $algo);

    $segments = array();
    $segments[] = JWT::urlsafeB64Encode(JWT::jsonEncode($header));
    $segments[] = JWT::urlsafeB64Encode(JWT::jsonEncode($payload));

```

```

        $signing_input = implode('.', $segments);

        $signature = JWT::sign($signing_input, $key, $algo);
        $segments[] = JWT::urlsafeB64Encode($signature);

        return implode('.', $segments);
    }

    /**
     * Sign a string with a given key and algorithm.
     *
     * @param string $msg    The message to sign
     * @param string $key    The secret key
     * @param string $method The signing algorithm. Supported
     *                      algorithms are 'HS256', 'HS384' and 'HS512'
     *
     * @return string        An encrypted message
     * @throws DomainException Unsupported algorithm was specified
     */
    public static function sign($msg, $key, $method = 'HS256')
    {
        $methods = array(
            'HS256' => 'sha256',
            'HS384' => 'sha384',
            'HS512' => 'sha512',
        );
        if (empty($methods[$method])) {
            throw new DomainException('Algorithm not supported');
        }
        return hash_hmac($methods[$method], $msg, $key, true);
    }

    /**
     * Decode a JSON string into a PHP object.
     *
     * @param string $input JSON string
     *
     * @return object        Object representation of JSON string
     * @throws DomainException Provided string was invalid JSON
     */
    public static function jsonDecode($input)
    {
        $obj = json_decode($input);
        if (function_exists('json_last_error') && $errno = json_last_error(
    )) {
            JWT::_handleJsonError($errno);
        } else if ($obj === null && $input !== 'null') {
            throw new DomainException('Null result with non-null input');
        }
    }

```



```

    }
    return $obj;
}

/**
 * Encode a PHP object into a JSON string.
 *
 * @param object|array $input A PHP object or array
 *
 * @return string          JSON representation of the PHP object or array
 *
 * @throws DomainException Provided object could not be encoded to valid JSON
 */
public static function jsonEncode($input)
{
    $json = json_encode($input);
    if (function_exists('json_last_error') && $errno = json_last_error()) {
        JWT::_handleJsonError($errno);
    } else if ($json === 'null' && $input !== null) {
        throw new DomainException('Null result with non-null input');
    }
    return $json;
}

/**
 * Decode a string with URL-safe Base64.
 *
 * @param string $input A Base64 encoded string
 *
 * @return string A decoded string
 */
public static function urlsafeB64Decode($input)
{
    $remainder = strlen($input) % 4;
    if ($remainder) {
        $padlen = 4 - $remainder;
        $input .= str_repeat('=', $padlen);
    }
    return base64_decode(strtr($input, '-_', '+/'));
}

/**
 * Encode a string with URL-safe Base64.
 *
 * @param string $input The string you want encoded
 */

```

```

    * @return string The base64 encode of what you passed in
    */
    public static function urlsafeB64Encode($input)
    {
        return str_replace('=', '', strtr(base64_encode($input), '+/', '-
_'));
    }

    /**
     * Helper method to create a JSON error.
     *
     * @param int $errno An error number from json_last_error()
     *
     * @return void
     */
    private static function _handleJsonError($errno)
    {
        $messages = array(
            JSON_ERROR_DEPTH => 'Maximum stack depth exceeded',
            JSON_ERROR_CTRL_CHAR => 'Unexpected control character found',
            JSON_ERROR_SYNTAX => 'Syntax error, malformed JSON'
        );
        throw new DomainException(
            isset($messages[$errno])
            ? $messages[$errno]
            : 'Unknown JSON error: ' . $errno
        );
    }
}
}

```

### Tercer paso:

Para probar nuestro sistema de autenticación crearemos un archivo Login.php en la carpeta controllers. Declaramos una clase login que extiende de REST\_Controller. Pero antes de declarar la clase colocamos los headers para nuestro servicio.

```

<?php
defined('BASEPATH') or exit('No direct script access allowed');
require APPPATH . 'libraries/REST_Controller.php';
require APPPATH . 'libraries/Format.php';
header('Access-Control-Allow-Origin: *');
header('Access-Control-Allow-Headers: X-API-KEY, Origin, X-Requested-
With, Content-Type, Accept, Access-Control-Request-Method, Authorization');
header('Access-Control-Allow-Methods: GET, POST, OPTIONS, PUT, DELETE');

```

```
header('Allow: GET, POST, OPTIONS, PUT, DELETE');
class login extends REST_Controller
{
```

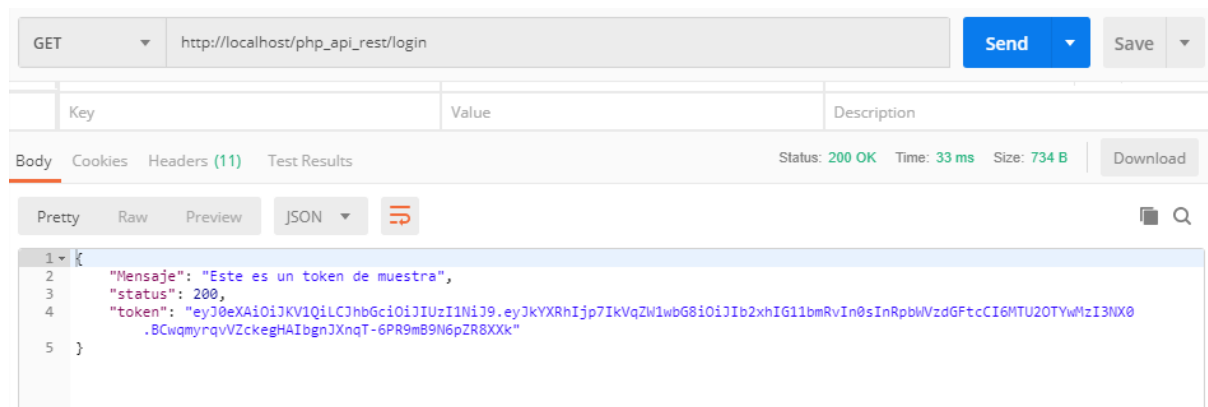
Nuestra clase login contiene un método constructor en el que cargamos los helpers necesarios para el manejo del token.

```
public function __construct()
{
    parent::__construct();
    // Cargamos el helper para poder crear el token
    $this->load->helper(['jwt', 'authorization', 'date']);
}
```

Para probar la generación del token crearemos un método de prueba. El token contiene en su payload (carga útil) la data y el tiempo de creación del token. Ten en cuenta que si no has agregado el helper de “date” en el archivo autoload.php, la función now() no funcionará.

```
public function index_get()
{
    // Creación del token de muestra
    $token['data'] = ['Ejemplo' => 'Hola mundo'];
    $token['timestamp'] = now();
    $tokenGenerated = AUTHORIZATION::generateToken($token);
    // Seteamos el código de estado HTTP 200
    $status = parent::HTTP_OK;
    $response = ['Mensaje' => 'Este es un token de muestra', 'status' => $status, 'token' => $tokenGenerated];
    // REST_Controller provee el siguiente método para enviar las respuestas
    $this->response($response, $status);
}
```

Si probamos este método desde la herramienta POSTMAN obtenemos el siguiente resultado:



Puedes validar el token y obtener el resultado del token descriptado. Y puedes acceder al contenido por medio de \$token->data o de \$token->timestamp.

```
//Validamos el token
$token = AUTHORIZATION::validateToken($tokenGenerated);
$response = ['Mensaje' => 'Este es un token de muestra', 'status' => $status,
'token' => $token];
```

Ahora que ya hemos visto como generar un token de prueba, lo haremos haciendo uso de las credenciales de un usuario, a partir de su nombre de usuario. Para ello, crearemos un método para validar que las credenciales correspondan con un registro de la tabla usuario. Este método recibe el nombre de usuario y la contraseña. Si un usuario coincide con las credenciales devuelve “true”, si no devuelve “false”.

```
public function validate_user($nombre_usuario, $contrasenia)
{
    try {
        $row = $this->db->get_where('usuario', array('nombre_usuario' => $nombre_usuario, 'contrasenia'
=> $contrasenia))->num_rows();
        if ($row === 1) {
            return true;
        } else {
            return false;
        }
    } catch (Exception $e) {
        $this->response(['Mensaje' => 'Ocurrió un error inesperado en el servidor'], parent:
:HTTP_INTERNAL_SERVER_ERROR);
    }
}
```

```
}  
}
```

Ahora sí, crearemos un método llamado `login_post()` que nos permita loguearnos y generar un token con las credenciales recibidas por medio del método POST. Lo primero que haremos será extraer las credenciales:

```
$username = $this->post('nombre_usuario');  
$contrasenia = $this->post('contrasenia');
```

Luego haremos uso del método recién creado para validar que esas credenciales pertenecen a un usuario de nuestra base de datos.




```
$data = $this->validate_user($username, $contrasenia);
```

Si recordamos, este método nos devolverá “true” (si existe un registro) o “false” (si no hay coincidencia), por lo tanto, evaluamos el valor de `$data`.

```
if ($data === true) {  
    // Creamos un token con los datos del usuario y enviamos la respuesta  
    $token['data'] = ['nombre_usuario' => $username];  
    $token['timestamp'] = now();  
    $tokenGenerated = AUTHORIZATION::generateToken($token);  
    // Seteamos el código de estado HTTP 200 para confirmar el éxito en la operación  
    $status = parent::HTTP_OK;  
    $response = ['Estado' => $status, 'token' => $tokenGenerated, 'Mensaje' => 'Token generado con éxito'];  
    // Enviamos la respuesta  
    $this->response($response, $status);  
}  
// Si el método validate_user nos devuelve false significa que no existe un usuario con esas credenciales  
else if ($data === false) {  
    $this->response(['Mensaje' => 'Usuario y/o contraseña inválidos'], parent::HTTP_NOT_FOUND);  
}
```

Probemos este método y veamos si nos crea el token para el registro que hemos ingresado a la base de datos. Siempre haciendo uso de POSTMAN.

Opciones

	id_usuario	nombre_usuario	contrasenia
<input type="checkbox"/>  Editar  Copiar  Borrar	1	dani	12345

---

http://localhost/php\_api\_rest/login/login

POST http://localhost/php\_api\_rest/login/login Send Save


Params Authorization Headers (1) **Body** Pre-request Script Tests Cookies Code Comments

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary JSON (application/json) Beat

```
1 {
2   "nombre_usuario": "dani",
3   "contrasenia": "12345"
4 }
```

---

Body Cookies Headers (11) Test Results Status: 200 OK Time: 44 ms Size: 734 B Download

Pretty Raw Preview JSON 

```
1 {
2   "Estado": 200,
3   "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhIjpbIm5vbWJyZV91c3Vhcm1vIjo1ZGFuSj9LCj0aW1lc3RhbXAiOjE1Njk2MDQ0OTZ9.fFo0MTrd1hDXST4LyPbP1DvvAgwgj26TzDzxoYz1Zo",
4   "Mensaje": "Token generado con éxito"
5 }
```

Ahora veamos qué pasa si ingresamos credenciales incorrectas:

POST http://localhost/php\_api\_rest/login/login Send Save


Params Authorization Headers (1) **Body** Pre-request Script Tests Cookies Code Comments

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary JSON (application/json) Beat

```
1 {
2   "nombre_usuario": "fake",
3   "contrasenia": "fake"
4 }
```

---

Body Cookies Headers (11) Test Results Status: 404 Not Found Time: 41 ms Size: 569 B Download

Pretty Raw Preview JSON 

```
1 {
2   "Mensaje": "Usuario y/o contraseña inválidos"
3 }
```

Vemos que nos lanza el error “Usuario y/o contraseña inválidos” ya que el método `validate_user()` ha devuelto `false`.

#### Cuarto paso:

Ya que hemos generado el token debemos validar que los métodos para los servicios de libro y genero se ejecuten siempre y cuando el token sea válido.

Nuestro método `index_get()` para la tabla género es el siguiente:

```
public function index_get($id = null)
{
    if (!empty($id)) {
        $data = $this->db-
>get_where("genero", ['id_genero' => $id])->row_array();
        if ($data == null) {
            $this-
>response(['Mensaje' => 'No hay datos coincidentes al id: ' . $id], parent::HT
TP_NO_CONTENT);
        }
    } else {
        $data = $this->db->get("genero")->result();
    }
    $this->response($data, parent::HTTP_OK);
}
```

Si no se recibe ningún parámetro, el método devuelve todos los registros de la tabla. Si recibe un id, devuelve el registro que corresponde a ese id.

Lo que haremos será validar el token para que este método se pueda ejecutar. Para ello, obtendremos las cabeceras de la petición y asignamos el token a una variable.

```
$headers = $this->input->request_headers();
$token = $headers['Authorization'];
```

Luego enviamos este supuesto token al método estático `verify_request()` que creamos en el archivo `authorization_helper.php`.

```
$dataValidate = AUTHORIZATION::verify_request($token);
```

Éste nos devuelve un código de estado que puede ser HTTP\_FORBIDDEN si no pasa el método validateToken(), HTTP\_OK si la validación es correcta y HTTP\_UNAUTHORIZED si se captura un error en el proceso de verificación.

Por lo tanto, para poder ejecutar nuestro método index\_get() es necesario que la respuesta obtenida sea un código HTTP\_OK. Ahora bien, debemos comprobar que el tiempo de vida del token, especificado en el archivo jwt.php, no haya finalizado. Para ello hacemos uso del método validateTimestamp().

```
$token = AUTHORIZATION::validateTimestamp($token);
```

Si todo va bien, este método nos devolverá el token en forma de objeto y es acá donde podremos, con toda seguridad, ejecutar el funcionamiento del método index\_get(), sino nos devolverá “false” y daremos un mensaje de error indicando que el tiempo de la sesión (del token) ya ha finalizado.

```
if ($token != false) {
    if (is_object($token)) {
        if (!empty($id)) {
            $data = $this->db->
>get_where("genero", ['id_genero' => $id])->row_array();
            if ($data == null) {
                $this->
>response(['Mensaje' => 'No hay datos coincidentes al id: ' . $id], parent::HT
TP_NO_CONTENT);
            }
        } else {
            $data = $this->db->get("genero")->result();
        }
        $this->response($data, parent::HTTP_OK);
    }
    //Si el tiempo de vida de el token ha finalizado mandamos un m
ensaje de error
} else if ($token === false) {
    $this->
>response(['Mensaje' => 'El tiempo de la sesión ha finalizado'], parent::HTTP_
FORBIDDEN);
}
```

A continuación, se presenta el método completo, con las respectivas validaciones:

```
public function index_get($id = null)
{
    // Obtenemos todos los headers de la solicitud y lo asignamos a una va
riable
```



```

$headers = $this->input->request_headers();
$token = $headers['Authorization'];
//Validamos el contenido del token
$dataValidate = AUTHORIZATION::verify_request($token);
//Si el token es válido comprobamos que el tiempo de vida de el token
no hay finalizado
if ($dataValidate === parent::HTTP_OK) {
    $token = AUTHORIZATION::validateTimestamp($token);
    //Si el token es válido ejecutamos el servicio
    if ($token != false) {
        if (is_object($token)) {
            if (!empty($id)) {
                $data = $this->db->
>get_where("genero", ['id_genero' => $id])->row_array();
                if ($data == null) {
                    $this->
>response(['Mensaje' => 'No hay datos coincidentes al id: ' . $id], parent::HT
TP_NO_CONTENT);
                }
            } else {
                $data = $this->db->get("genero")->result();
            }
            $this->response($data, parent::HTTP_OK);
        }
        //Si el tiempo de vida de el token ha finalizado mandamos un m
ensaje de error
    } else if ($token === false) {
        $this->
>response(['Mensaje' => 'El tiempo de la sesión ha finalizado'], parent::HTTP_
FORBIDDEN);
    }
    } else if ($dataValidate === parent::HTTP_UNAUTHORIZED) {
        $this->
>response(['Mensaje' => 'No tienes acceso a este servicio'], $dataValidate);
    } else if ($dataValidate === parent::HTTP_FORBIDDEN) {
        $this->
>response(['Mensaje' => 'El token de acceso es incorrecto'], $dataValidate);
    }
}
}

```

Para finalizar haremos las pruebas correspondientes con la herramienta POSTMAN ejecutando el método `index_get()` de la tabla genero.


- Si no especificamos la cabecera Authorization en la solicitud.

GET ▼ http://localhost/php\_api\_rest/api/genero/ Send ▼

Params Authorization **Headers (1)** Body Pre-request Script Tests Cookies C

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	Content-Type	application/json			
	Key	Value	Description		

Body Cookies **Headers (11)** Test Results Status: 401 Unauthorized Time: 27 ms Size: 1.33 KB

Pretty Raw Preview JSON ▼ 

```
1 {  
2   "Mensaje": "No tienes acceso a este servicio"  
3 }
```


- Si especificamos la cabecera Authorization con un token inválido.

GET ▼ http://localhost/php\_api\_rest/api/genero/ Send ▼

Params Authorization **Headers (2)** Body Pre-request Script Tests Cookies Codi

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	Content-Type	application/json			
<input checked="" type="checkbox"/>	Authorization	soy un token			
	Key	Value	Description		

Body Cookies **Headers (11)** Test Results Status: 403 Forbidden Time: 46 ms Size: 566 B

Pretty Raw Preview JSON ▼ 

```
1 {  
2   "Mensaje": "El token de acceso es incorrecto"  
3 }
```

- Si el token es válido obtenemos los recursos.

http://localhost/php\_api\_rest/api/genero/

GET http://localhost/php\_api\_rest/api/genero/ Send

Params Authorization Headers (2) Body Pre-request Script Tests Cookies Code

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	Content-Type	application/json			
<input checked="" type="checkbox"/>	Authorization	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhbj...			
	Key	Value	Description		

Body Cookies Headers (11) Test Results Status: 200 OK Time: 38 ms Size: 666 B

Pretty Raw Preview JSON

```
1 [
2   {
3     "id_genero": "1",
4     "titulo": "Comedia"
5   },
6   {
7     "id_genero": "2",
8     "titulo": "Fantasía"
9   },
10  {
11    "id_genero": "3",
12    "titulo": "Tragedia"
13  },
14  {
15    "id_genero": "4",
16    "titulo": "Romance"
17  }
18 ]
```

- Si el tiempo de vida del token ha finalizado.

http://localhost/php\_api\_rest/api/genero/

GET http://localhost/php\_api\_rest/api/genero/ Send

Params Authorization Headers (2) Body Pre-request Script Tests Cookies Code

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	Content-Type	application/json			
<input checked="" type="checkbox"/>	Authorization	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJkYXRhbj...			
	Key	Value	Description		

Body Cookies Headers (11) Test Results Status: 403 Forbidden Time: 32 ms Size: 571 B

Pretty Raw Preview JSON

```
1 {
2   "Mensaje": "El tiempo de la sesión ha finalizado"
3 }
```

## **Bibliografía**

Munshi, R. (27 de abril de 2019). *NewCodingEra*. Obtenido de NewCodingEra:  
<https://newcodingera.com/jwt-in-codeigniter/>

ParitoshVaidya. (14 de septiembre de 2019). *Github*. Obtenido de Github:  
<https://github.com/ParitoshVaidya/CodeIgniter-JWT-Sample>