

# JWT Symfony

Especialidad PHP



Felipe Izquierdo Romero

## Índice

<u>1</u>	<u>Descripción del problema</u>
<u>2</u>	<u>Desarrollo</u>

## 1. Descripción del problema

Implementar JWT para una aplicación de gestión de productos en Symfony. Permitiendo el registro y el login de usuarios y dejando únicamente a usuarios conectados poder acceder a los recursos de la aplicación.

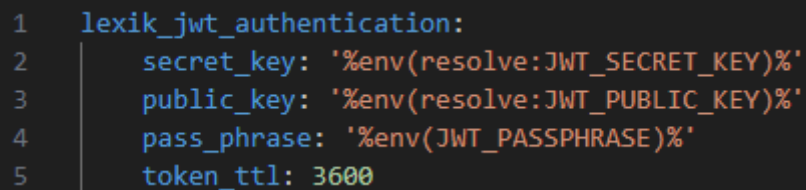
## 2. Desarrollo

Para este lab reutilizando el código de ejercicios anteriores creo un nuevo contenedor con el contenido de el lab anterior.

Primero hay que instalar el paquete JWT y crear la clave SSH con los comandos.

```
composer require lexik/jwt-authentication-bundle  
php bin/console lexik:jwt:generate-keypair
```

Posteriormente en el archivo “lexik\_jwt\_authentication.yaml” al final del todo añadimos un tiempo de validez del token con la línea “token\_ttl:3600”.



```
1 lexik_jwt_authentication:  
2     secret_key: '%env(resolve:JWT_SECRET_KEY)%'  
3     public_key: '%env(resolve:JWT_PUBLIC_KEY)%'  
4     pass_phrase: '%env(JWT_PASSPHRASE)%'  
5     token_ttl: 3600
```

*Ilustración 1- Tiempo de validez del token*

Ahora habría que pasar a crear la entidad usuarios con el comando:

```
php bin/console make:user
```

Importante es hacer que la entidad implemente de “UserInterface” he implementar los métodos “getUserIdentifier”, “getRoles”, “getPassword”, “getSalt”, “eraseCredentials”, “getUsername”.

Tras la creación de la entidad y repositorio con el comando anterior pasamos a crear el controlador.

Definimos cuatro atributos importantes para trabajar con los datos y los asignamos en el constructor.

```

14  class UserController extends AbstractController
15  {
16      private $userProvider;
17      private $passwordHasher;
18      private $entityManager;
19      private $JWTTokenManager;
20
21      public function __construct(
22          UserProviderInterface $userProvider,
23          UserPasswordHasherInterface $passwordHasher,
24          EntityManagerInterface $entityManager,
25          JWTTokenManagerInterface $JWTTokenManager)
26      {
27          $this->userProvider = $userProvider;
28          $this->passwordHasher = $passwordHasher;
29          $this->entityManager = $entityManager;
30          $this->JWTTokenManager = $JWTTokenManager;
31      }

```

*Ilustración 2- Atributos y Constructor UserContrller.php*

Y creamos los dos métodos register y login. Ambos devuelven un token para poder acceder a los recursos de la web.

```

33  public function register(Request $request)
34  {
35      $user = new User();
36
37      $user->setEmail($request->get('email'));
38      $user->setPassword($this->passwordHasher->hashPassword($user, $request->get('password')));
39
40      $this->entityManager->persist($user);
41      $this->entityManager->flush();
42
43      $token = $this->JWTTokenManager->create($user);
44
45      return $this->json(['message' => 'Usuario creado con éxito', 'token' => $token], Response::HTTP_CREATED);
46  }

```

*Ilustración 3- Funcion Register*

```

48  public function login(Request $request)
49  {
50      $user = $this->userProvider->loadUserByIdentifier($request->get('email'));
51
52      if(!$user || !$this->passwordHasher->isPasswordValid($user,$request->get('password')))
53      {
54          return $this->json(['message' => 'Credenciales incorrectos'],Response::HTTP_BAD_REQUEST);
55      }
56
57      $token = $this->JWTTokenManager->create($user);
58
59      return $this->json(['token' => $token]);
60  }

```

*Ilustración 4- Funcion Login*

A continuación se deben configurar las rutas en el archivo “routes.yaml” .

```
26 app_register:
27   path: '/register'
28   controller: 'App\Controller\UserController::register'
29   methods: ['POST']
30
31 app_login:
32   path: '/login'
33   controller: 'App\Controller\UserController::login'
34   methods: ['POST']
```

*Ilustración 5- Rutas Register y Login*

Para finalizar se debe configurar el archivo “security.yaml” primero añadiendo la ruta de login como de la API dentro de firewalls.

```
18 login:
19   pattern: ^/login
20   stateless: true
21   provider: app_user_provider
22   form_login:
23     username_parameter: email
24     password_parameter: password
25
26 api:
27   pattern: ^/
28   stateless: true
29   jwt: ~
```

*Ilustración 6- Rutas de login y API*

Después los accesos permitidos según que autenticación necesaria escribiendo de menos restrictivo a más restrictivo.

```
45 access_control:
46   - { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }
47   - { path: ^/register, roles: IS_AUTHENTICATED_ANONYMOUSLY }
48   - { path: ^/, roles: IS_AUTHENTICATED_FULLY }
```

*Ilustración 7- Restricciones*

Con esto ya tendríamos una aplicación que permitiese registrar y logear usuarios además de compartir sus recursos solamente a usuarios logeados.