# Generar Entities y bundle

## Pasos para crear Bundles

Para crear bundles se usa el comando

## php bin/console generate:bundle --namespace=BackendBundle --format=yml

## Pasos para crear Entities

Para crear entities, con este comando se crea un fichero yml con las configuraciones en formato YML

Php bin/console doctrine:mapping:import BackendBundle yml

Generara un archivo yml por cada tabla de la DB. En la ruta bundle/condig/doctrine/

Para crear entities en solo una tabla se crea con el comando

Php bin/console doctrine:mapping:import BackendBundle yml –filter =”NombreTabla”

Para generar las entidades se lanza el comando

Php bin/console doctrine:generate:entities BackendBundle

## Pasos para probar un controlador y la carga de datps

En default controller utilizar lo siguiente:

/\*\*

\* @Route("/pruebas", name="pruebas")

\*/

public function pruebasAction(Request $request)

{

$em = $this->getDoctrine()->getManager();

$users=$em->getRepository('BackendBundle:User')->findAll();

var\_dump($users);

die();

}

# Instalar paquetes con Composer

## Installar JWT y el paginador

Agregar en composer.json:

"require": {

"php": ">=5.5.9",

"symfony/symfony": "3.0.\*",

"doctrine/orm": "^2.5",

"doctrine/doctrine-bundle": "^1.6",

"doctrine/doctrine-cache-bundle": "^1.2",

"symfony/swiftmailer-bundle": "^2.3",

"symfony/monolog-bundle": "^2.8",

"sensio/distribution-bundle": "^5.0",

"sensio/framework-extra-bundle": "^3.0.2",

"incenteev/composer-parameter-handler": "^2.0",

"firebase/php-jwt":"^3.0.0",

"knplabs/knp-paginator-bundle": "2.5.\*"

},

Ejecutar el comando:

composer update

# Rutas YML

## Cambiar las rutas para gestionarlas desde otro método con YML

Crear carpeta en AppBundle que se llame resources

Crear en esta carpeta Resources una carpeta llamada config

Crear archivo routing.yml

Crear un directorio llamado routing

Crear archivo llamado default.yml

-----------------------

Modificar en ROOT app/config/routing.yml

app:

resource: "@AppBundle/Resources/config/routing.yml"

prefix: /

-----------------------

en AppBundle Resource config routing.yml

app\_default:

resource: "@AppBundle/Resources/config/routing/default.yml"

prefix: /

-----------------------

En default.yml

default\_index:

path: /

defaults: {\_controller: "AppBundle:Default:index"}

methods: GET

pruebas\_index:

path: /pruebas

defaults: {\_controller: "AppBundle:Default:pruebas"}

methods: GET

# Funciones

## Crear una función para convertir un array en JSON

El controlador que devolverá los datos quedará de la forma:

public function pruebasAction(Request $request)

{

$em = $this->getDoctrine()->getManager();

$users=$em->getRepository('BackendBundle:User')->findAll();

$pruebas= array("id"=>1, "nombre"=>"andes");

return $this->json($users);

}

Esto se realizará ya que siempre se devolverá un JSON en cada método,. La Funcion que convertirá todo a un JSON para usar en un controller es:

public function json($data){

$normalizers = array( new \Symfony\Component\Serializer\Normalizer\GetSetMethodNormalizer());

$encoders= array ("json"=>new \Symfony\Component\Serializer\Encoder\JsonEncode());

$serializer= new \Symfony\Component\Serializer\Serializer($normalizers, $encoders);

$json = $serializer->serialize($data, 'json');

$response = new \Symfony\Component\HttpFoundation\Response();

$response->setContent($json);

$response->headers->set("Content-Type", "application/json");

return $response;

}

# SERVICIOS

Se crearán servicios adicionales en el resto del código por lo cual a continuación se describirá uno de ellos, cada capitulo tendrán otros mas.

## Crear un servicio Helpers para convertir un objeto o un array a JSON.

Crear un directorio en el bundle que se llame: services

Crear en esta carpeta un archivo clase php que se llame: helpers.php

Declarar el archivo services:

<?php

namespace AppBundle\Services;

class Helpers {

}

Copiar la función del controlador al servicio.

Ir a app/config/services.yml y editar el archivo:

parameters:

# parameter\_name: value

**services:**

# service\_name:

# class: AppBundle\Directory\ClassName

# arguments: ["@another\_service\_name", "plain\_value", "%parameter\_name%"]

**app.helpers:**

**class: AppBundle\Services\Helpers**

**arguments: ["null"]**

Modificamos el controlador para invocar el servicio

public function pruebasAction(Request $request)

{

**$helpers = $this->get("app.helpers");**

$em = $this->getDoctrine()->getManager();

$data=$em->getRepository('BackendBundle:User')->findAll();

**return $helpers->json($data);**

}

# Crear LOGIN para servicio API

## Primeros pasos para Login

### Ruta

Crear una ruta en el directorio default.yml con el acceso al login

default\_login:

path: /login

defaults: {\_controller: "AppBundle:Default:login"}

methods: POST

### Controlador

En el encabezado adjuntar la siguiente linea

use Symfony\Component\Validator\Constraints as Assert;

ge

Crear el contrenido en el controlador en donde se reciva por POST los datos de login en JSON, a continuacion se valida si se recibe un JSON con data y a continuacion se decodifica. la validación de mail se hace con email Constraint, de la siguiente manera,

public function loginAction(Request $request){

$helpers = $this->get("app.helpers");

$json= $request->get("json", null);

if ($json!=null){

$params=json\_decode($json);

$email= (isset($params->email))? $params->email : null;

$password=(isset($params->password))? $params->password : null;

$emailCons= new Assert\Email();

$emailCons->message="Formato de email no valido";

$validate\_email= $this->get("validator")->validate($email,$emailCons);

if(count($validate\_email)==0 && $password!=null){

echo "Data Success!!!";

} else {

echo "Data Incorrect!!!!!";

}

} else {

echo "Not received, Please send JSON data to login";

die();

}

}

Se debe probar el servicio con POSTMAN enviando un JSON con los datos Creados asi:

Key :json value: {"email": "andes@gmail.com", "password": "1234"}

## Crear Token para Login

Para realizar el Login de la API se implementara JWT para crear los tokens de los usuarios registrados y pasar este token entre los sitios a los cuales se tendrá acceso.

Crear Nuevo servicio JWT para validar Usuario y tambien para codificar y decodificar TOKENS

### Services.yml

Se creara un servicio llamado jwt\_auth en el directorio de los servicios de app/config/services.yml , este servicio tendra como argumentos el manejo del ORM doctrine y su entity manager,

app.jwt\_auth:

class: AppBundle\Services\JwtAuth

arguments: ["@doctrine.orm.entity\_manager"]

### JwtAuth.php

En la carpeta de servicios del AppBundle se creara una clase php con nombre jwtAth.php con el siguiente codigo:

<?php

namespace AppBundle\Services;

use Firebase\JWT\JWT;

class JwtAuth {

public $manager;

public function \_\_construct ($manager){

$this->manager=$manager;

}

public function signup($email, $password, $getHash=NULL){

$key="clave-secreta";

$user =$this->manager->getRepository('BackendBundle:User')->findOneBy(

array (

"email"=>$email,

"password"=>$password

)

);

$signup=false;

if(is\_object($user)){

$signup=true;

}

if($signup==true){

// return array("status"=>"success", "data"=>"Correct Login!!!");

//Datos para crear el token

$token= array(

"sub"=>$user->getId(),

"email"=>$user->getEmail(),

"name"=>$user->getName(),

"surname"=>$user->getSurname(),

"password"=>$user->getPassword(),

"image"=>$user->getImage(),

"iat"=>time(),

"exp"=>time()+( 7 \* 24 \* 60 \*60 )

);

$jwt= JWT::encode($token,$key, 'HS256');

$decoded=JWT::decode($jwt,$key, array("HS256"));

if($getHash!=null){

return $jwt;

}else {

return $decoded;

}

}

else{

return array("status"=>"error", "data"=>"Login Failed, User and Password incorrect!!!");

die();

}

}

}

### Controlador

En los encabezados del controlador se utilizara el componente JSON response:

use Symfony\Component\HttpFoundation\JsonResponse as JsonResponse;

Se creara un metodo denominado LoginAction con el siguiente codigo:

**public function loginAction(**Request $request){

$helpers = $this->get("app.helpers");

$jwt\_auth = $this->get("app.jwt\_auth");

$json= $request->get("json", null);

if ($json!=null){

$params=json\_decode($json);

$email= (isset($params->email))? $params->email : null;

$password=(isset($params->password))? $params->password : null;

$getHash=(isset($params->gethash))? $params->gethash : null;

$emailCons= new Assert\Email();

$emailCons->message="Formato de email no valido";

$validate\_email= $this->get("validator")->validate($email,$emailCons);

if(count($validate\_email)==0 && $password!=null){

if($getHash== null){

$signup=$jwt\_auth->signup($email,$password);

}

else{

$signup=$jwt\_auth->signup($email,$password, true);

}

return new JsonResponse($signup);

} else {

return $helpers->json(array(

"status" =>"error",

"data"=>"API login Not Valid"

));

}

}

### Pruebas

Se realizaran las pruebas con la herramienta postman pasando argumentos por el metodo POST de la siguiente manera:

<http://localhost/SymAPI/web/app_dev.php/login>

los datos pasados como llave, valor para que devuelva un Token son:

key: json

value: {"email":"admin@gmail.com","password":"admin","gethash":"true"}

Los datos pasados para que devuelva un JSON con los datos del Usuario son:

key: json

value: {"email":"admin@gmail.com","password":"admin”}

# Crear Administracion de usuarios (Controller, Rutas,Servicios)

Se desarrollara un controlador para manejar la tarea de usuarios (CRUD), inicialmente se crearan los registros en la table validando si se encuentran repetidos.

Incialmente se agregara al archive de rutas del controlador el direccionamiento de las rutas para el manejo de usuarios

app\_user:

resource: "@AppBundle/Resources/config/routing/user.yml"

prefix: /user

en el archive user.yml se gestiionaran las rutas relaciondas con la administracion de usuarios

## Metodo para crear usuarios user New

Antes de comenar se parametrizara la ruta new en el archive de rutas user.yml la ruta para crear usuarios que se manejara por el metodo POST y en el controlado User.

user\_new:

path: /new

defaults: {\_controller: "AppBundle:User:new"}

methods: POST

Se cerara el método newAction para la modificacion de los usuarios,

Se cargara el servicio helpers , y se realizaran las validaciones al archive JSON para verificar que los datos recibidos tengan la integridad y tipado necesario.

Si el archivo JSON no es válido o si los datos no existen o están en blanco no se realizará la carga de los datos.

public function newAction(Request $request){

$helpers = $this->get("app.helpers");

$json=$request->get("json", null);

$params=json\_decode($json);

$data= array();

if($json!=null){

$createdAt= new \Datetime("now");

$image= null;

$role= "user";

$email =(isset($params->email))? $params->email: null;

$name =(isset($params-> name))? $params->name: null;

$surname =(isset($params->surname))? $params->surname: null;

$password =(isset($params->password))? $params->password: null;

$emailCons= new Assert\Email();

$emailCons->message="Formato de email no valido";

$validate\_email= $this->get("validator")->validate($email,$emailCons);

if($email != null && count ($validate\_email)==0 && $password != null && $name != null && $surname != null)

{

$user = new User();

$user->setCreatedAt($createdAt);

$user->setEmail($email);

$user->setRoles($role);

$user->setName($name);

$user->setSurname($surname);

$user->setPassword($password);

$user->setImage($image);

$em = $this->getDoctrine()->getManager();

$isset\_user = $em->getRepository("BackendBundle:User")->findBy(

array (

"email"=>$email

));

if(count($isset\_user)==0){

$em->persist($user);

$em->flush();

$data ["status"]='success';

$data["msg"]="New User was created";

$data["code"]=200;

} else {

$data= array (

"status"=>"error",

"code"=>400,

"msg"=>"User was not created, user already exist!"

) ;

}

}

$data = array (

"status"=>"error",

"code"=>400,

"message"=>"JSON data is invalid"

);

} else {

$data = array (

"status"=>"error",

"code"=>400,

"message"=>"User not created, JSON data is Null"

);

}

return $helpers->json($data);

die();

}

### Pruebas de creacion de usuarios

La URL de pruebas es:

<http://localhost/SymAPI/web/app_dev.php/user/new>

Se deberá enviar el objeto JSON por método POST con los siguientes datos:

Key: json

Value: {"email":"user@gmail.com", "password":"user","name":"user","surname":"user surname"}

## Editar Usuarios:

Antes de iniciar con la edición de usuarios, se empleara un servicio que tomará una variable llamada authorization con la cual se validara la identidad de la persona que envía el token y se realizará las tareas de ejecución de los servicios en el API esto lo Haran dos servicios que seran CheckTokens yAuthCheck.

### Servicio checkTokens

Se creará en jwt una function que valide el token y devuelva un resultado true si el token fue validado, de lo contrario será false. Tambien se creará la function de devolver los valores del Token si se desean obtener sus valores.

Servicio Jwt.php

Se modificara la case del servicio creando un constructor para crear la clave que sera utilizada en todos los servicios adicionales:

class JwtAuth {

public $manager;

public $key;

public function \_\_construct($manager) {

$this->manager = $manager;

$this->key = "clave-secreta";

}

}

El service que validará los datos tendra el siguiente codigo:

public function checkToken($jwt, $getIdentity = false) {

$key = $this->key;

$auth = false;

try {

$decoded = JWT::decode($jwt, $key, array("HS256"));

} catch (\UnexpectedValueException $e) {

$auth = false;

} catch (\DomainException $e) {

$auth = false;

}

if ($decoded->sub) {

$auth = true;

} else {

$auth = false;

}

// IF Parameter 2 is true

if ($getIdentity == true) {

return $decoded;

} else {

return $auth;

}

}

### Servicio AuthCheck

Helpers.php

Este servicio fue ajustado para que funcionara desde Helpers.php por lo cual en helpers se invoca de la siguiente forma:

Se crea un constructor con los datos del servicio jwt\_auth para ser invocado desde este otro servicio:

**class Helpers {**

public $jwt\_auth;

public function \_\_construct($jwt\_auth) {

$this->jwt\_auth = $jwt\_auth;

}

A continuación se crea un servicio que realizara tareas de validacion de tokens desde helpers.php

**public function authCheck($hash, $getIdentity = false) {**

$jwt\_auth = $this->jwt\_auth;

$auth = false;

if ($hash != null) {

if ($getIdentity == false) {

$check\_token = $jwt\_auth->checkToken($hash);

if ($check\_token == true) {

$auth = true;

}

} else {

$check\_token = $jwt\_auth->checkToken($hash, true);

if (is\_object($check\_token)) {

$auth = $check\_token;

}

}

}

return $auth;

}

Finalmente se debe modificar lso argumentos del archvio services.yml de la sgiguiente forma:

services:

app.jwt\_auth:

class: AppBundle\Services\JwtAuth

arguments: ["@doctrine.orm.entity\_manager"]

app.helpers:

class: AppBundle\Services\Helpers

arguments: ["@app.jwt\_auth"]

### Controlador de pruebas: pruebas Controller

Se modificara leven,emte el codigo del pruebasController para validar el funcionamiento del servicio, las rutas tambien deberan ser modificadas para aceptar parametros porPOST.

public function pruebasAction(Request $request) {

$helpers = $this->get("app.helpers");

$hash = $request->get("authorization");

$data = $helpers->authCheck($hash, false);

var\_dump($data);

die();

}

A continuación se tendrá que enviar por postman una llave llamada authorization con un token valido, este podrá ser obtenido del proceso de Login que devuelve un token:

URL:

http://localhost/SymAPI/web/app\_dev.php/pruebas

**[POST]**

**Key:**

authorization

**Value:** \*\*\*TOKEN\*\*\*

### Editar Usuarios

Se creará unnn controller llamado edit Action, una ruta para aceptar lso edit de la siguiente forma:

Ruta: user.yml:

**user\_edit:**

path: /edit

defaults: {\_controller: "AppBundle:User:edit"}

methods: POST

Controlador: userController:

**public function editAction(Request $request) {**

$helpers = $this->get("app.helpers");

$hash = $request->get("authorization", null);

//Recibir token de validacion

$authCheck = $helpers->authCheck($hash);

if ($authCheck == true) {

//Decofificar los datos

$identity = $helpers->authCheck($hash, true);

$em = $this->getDoctrine()->getManager();

$user = $em->getRepository("BackendBundle:User")->findOneBy(

array(

"id" => $identity->sub

));

$json = $request->get("json", null);

$params = json\_decode($json);

$data = array(

"status" => "error",

"code" => 400,

"msg" => "Updated Failed created, JSON data is Null"

);

if ($json != null) {

$createdAt = new \Datetime("now");

$image = "image.jpg";

$role = "user";

$email = (isset($params->email)) ? $params->email : null;

$name = (isset($params->name)) ? $params->name : null;

$surname = (isset($params->surname)) ? $params->surname : null;

$password = (isset($params->password)) ? $params->password : null;

$emailCons = new Assert\Email();

$emailCons->message = "Formato de email no valido";

$validate\_email = $this->get("validator")->validate($email, $emailCons);

if ($email != null && count($validate\_email) == 0 && $name != null && $surname != null) {

$user->setCreatedAt($createdAt);

$user->setEmail($email);

$user->setRoles($role);

$user->setName($name);

$user->setSurname($surname);

if ($password != null) {

//Cifrar password

$pass = hash('sha256', $password);

$user->setPassword($pass);

}

$user->setImage($image);

$em = $this->getDoctrine()->getManager();

$isset\_user = $em->getRepository("BackendBundle:User")->findBy(

array(

"email" => $email

));

if (count($isset\_user) == 0 || $identity->email == $email) {

$em->persist($user);

$em->flush();

$data ["status"] = 'success';

$data["msg"] = " User was Updated";

$data["code"] = 200;

} else {

$data = array(

"status" => "error",

"code" => 400,

"msg" => "Updated user Failed, user already exist!"

);

}

}

}

return $helpers->json($data);

die();

} else {

$data = array(

"status" => "error",

"code" => 400,

"msg" => "Authorization failed"

);

}

}

Probar la edición de usuarios:

Para probar la edicion de usuarios se debe cargar una llave authorization y otra llave llamada json, como se hizo en el ejercico anterior de crear Usuario y validar token:

**URL:**

http://localhost/SymAPI/web/app\_dev.php/uer/edit

**[POST]**

**Key:**

authorization

**Value:** \*\*\*TOKEN\*\*\*

**Key:**

Json

**Value:**

{"email":"admin@3gmail.com","password":"admin","name":"admin edited ","surname":"admin edited"}