APUNTES SYMFONY 4.4

[Capte la atención del lector con una descripción breve atractiva. Esta descripción es un resumen breve del documento. Cuando esté listo para agregar contenido, haga clic aquí y empiece a escribir.]

[Subtítulo del documento]

Contenido

[1. INSTALACION. 4](#_Toc57631247)

[2. CONSOLA. 4](#_Toc57631248)

[3. CREAR NUEVO CONTROLADOR. 4](#_Toc57631249)

[4. RUTAS Y ACCIONES. 5](#_Toc57631250)

[4.1 PARAMETROS OBLIGATORIO EN RUTAS. 6](#_Toc57631251)

[4.2 PARAMETROS OPCIONALES EN RUTAS. 6](#_Toc57631252)

[4.2.1 PARAMETRO OPCIONAL CON VALOR POR DEFECTO. 6](#_Toc57631253)

[4.2.2 PARAMETRO OPCIONAL SIN VALOR POR DEFECTO. 6](#_Toc57631254)

[4.3 RUTAS AVANZADAS. 7](#_Toc57631255)

[4.4 MOSTRAR RUTAS POR CONSOLA. 7](#_Toc57631256)

[4.5 REDIRECCIONES. 7](#_Toc57631257)

[4.5.1 REDIRIGIR A UNA RUTA 7](#_Toc57631258)

[4.5.2 REDIRECCION 301. 8](#_Toc57631259)

[4.5.3 REDIRECCION CON PASO DE PARAMETROS. 8](#_Toc57631260)

[4.5.4 REDIRECCION A UNA URL. 9](#_Toc57631261)

[5. VISTAS Y MOTOR DE PLANTILLAS TWIG. 9](#_Toc57631262)

[5.1 PLANTILLA BASE. 9](#_Toc57631263)

[5.2 HEREDAR PLANTILLA BASE EN UNA VISTA. 11](#_Toc57631264)

[5.3 HEREDAR CONTENIDO DE LA PLANTILLA MASTER 12](#_Toc57631265)

[5.4 CREAR VARIABLES EN TWIG. 12](#_Toc57631266)

[5.5 CONCATENAR VARIABLES. 13](#_Toc57631267)

[5.6 ARRAYS EN TWIG. 13](#_Toc57631268)

[5.6.1 ARRAYS 13](#_Toc57631269)

[5.6.2 ARRAYS ASOCIATIVOS 13](#_Toc57631270)

[5.7 ESTRUCTURAS DEL CONTROL. 14](#_Toc57631271)

[5.7.1 IF – ELSE. 14](#_Toc57631272)

[5.7.2 BUCLES. 14](#_Toc57631273)

[5.7.3 STARTS-END. 15](#_Toc57631274)

[5.8 FUNCIONES PREDEFINIDAS EN TWIG. 16](#_Toc57631275)

[5.8.1 VALOR MINIMO Min(). 17](#_Toc57631276)

[5.8.2 VALOR MAXIMO Max(). 17](#_Toc57631277)

[5.8.3 ELEMENTO ALEATORIO DE UN ARRAY. 17](#_Toc57631278)

[5.8.4 RANGE(). 17](#_Toc57631279)

[5.9 INCLUDES. 18](#_Toc57631280)

[5.9.1 INCLUDE BASICO. 18](#_Toc57631281)

[5.9.2 INCLUDE CON PASO DE PARAMETROS. 18](#_Toc57631282)

[5.9.3 CLAUSULA with\_context = false 18](#_Toc57631283)

[5.10 FILTROS. 19](#_Toc57631284)

[5.10.1 LENGTH 19](#_Toc57631285)

[5.10.2 TRIM. 19](#_Toc57631286)

[5.10.3 ENCADENAR FILTROS. 19](#_Toc57631287)

[5.11 CREAR EXTENSIONES. 20](#_Toc57631288)

[6. BASES DE DATOS EN SYMFONY. 23](#_Toc57631289)

[6.1 CONFIGURACION DE LA BD. 23](#_Toc57631290)

[6.2 CREACION DE ENTIDADES DESDE LA BASE DE DATOS. 23](#_Toc57631291)

[6.3 GENERAR ENTIDADES DESDE SYMPONY. 24](#_Toc57631292)

[6.3.1 SOBRESCRIBIR ENTIDAD(Añadir registros). 25](#_Toc57631293)

[6.4 GENERACION DE TABLAS A PARTIR DE ENTIDADES. 25](#_Toc57631294)

[6.5 MODIFICAR ENTIDADES. 25](#_Toc57631295)

[6.6 GUARDAR EN LA BASE DE DATOS. 26](#_Toc57631296)

[6.7 CONSULTAS EN LA CONSOLA DE DOCTRINE. 26](#_Toc57631297)

[6.8 CONSULTAS ORM.OBTENER REGISTRO.find(). 27](#_Toc57631298)

[6.9 CONSULTAS ORM.LISTADO DE REGISTROS.findAll(). 27](#_Toc57631299)

[6.10 CONSULTAS ORM.CONSULTAS.TIPOS DE find(). 29](#_Toc57631300)

[6.10.1 OBTENER REGISTRO EN BASE A UNA CONDICION findOneBy(). 29](#_Toc57631301)

[6.10.2 CONSULTAS ORM.OBTENER VARIOS REGISTROS EN BASE A UNA CONDICION findBy(). 30](#_Toc57631302)

[6.10.3 CONSULTAS ORM.AÑADIR CLAUSULA ORDER BY. 30](#_Toc57631303)

[6.11 CONSULTA ORM.OBTENER OBJETO/REGISTRO AUTOMATICAMENTE. 31](#_Toc57631304)

[6.12 CONSULTAS ORM. ACTUALIZACION DE REGISTROS EN LA BD. 32](#_Toc57631305)

[6.13 ELIMINAR DE REGISTROS EN LA BD. 33](#_Toc57631306)

[6.14 QUERY BUILDER. 34](#_Toc57631307)

[6.14.1 CONDICIONES. 34](#_Toc57631308)

[6.14.2 ORDER BY. 35](#_Toc57631309)

[6.15 LENGUAJE DQL. 36](#_Toc57631310)

[6.16 LENGUAJE SQL. 37](#_Toc57631311)

[6.17 CREAR REPOSITORIOS. 38](#_Toc57631312)

[7. FORMULARIOS. 41](#_Toc57631313)

[7.1 CREAR FORMULARIOS. 41](#_Toc57631314)

[7.2 PERSONALIZAR CAMPOS. 42](#_Toc57631315)

[7.3 RECIBIR DATOS DEL FORMULARIO. 43](#_Toc57631316)

[7.4 VALIDACION DE CAMPOS. 45](#_Toc57631317)

[7.5 FORMULARIOS SEPARADOS EN CLASES. 46](#_Toc57631318)

[7.6 VALIDACION DE DATOS AISLADOS. 48](#_Toc57631319)

# 1. INSTALACION.

Crear nuevo proyecto:



Instalacion packete apache:



# 2. CONSOLA.

Ver listado de comandos:



# 3. CREAR NUEVO CONTROLADOR.



Nos genera tanto el controlador(**src/controller/HomeController.php**) como una vista asociada(**templates/home/home.html.twig**).

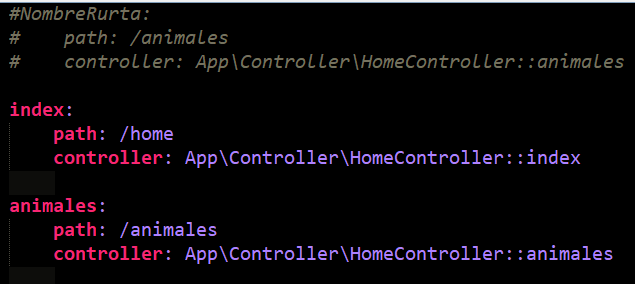
Controllador



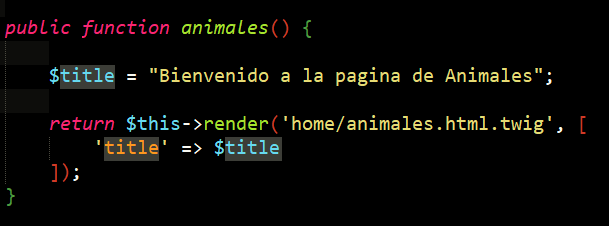
# 4. RUTAS Y ACCIONES.

Vamos a configurar para que las rutas estén en un fichero .yaml , en vez de como las usa por defecto.

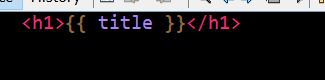
Para ello vamos a **config/routes/routes.yaml**, y las vamos a generar de la siguiente forma:



En el controlador HomeController definimos el método animales(), que nos devuelve una variable a la vista(**templates/home/animales.html.twig**)

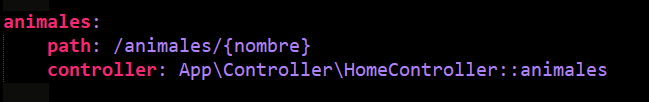


Vista

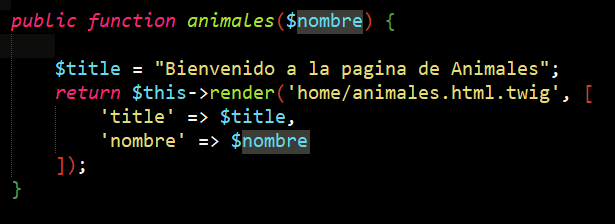


## 4.1 PARAMETROS OBLIGATORIO EN RUTAS.

Es bastante parecido a Laravel, en el fichero de rutas .yaml, vamos a añadir el **parámetro obligatorio nombre**



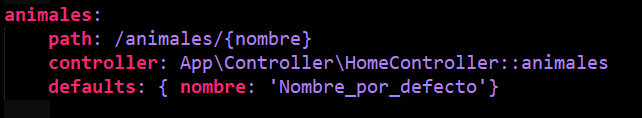
Vamos a recibir este parámetro(nombre), en el controlador animales():



## 4.2 PARAMETROS OPCIONALES EN RUTAS.

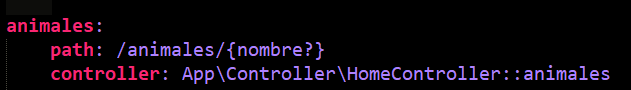
### 4.2.1 PARAMETRO OPCIONAL CON VALOR POR DEFECTO.

Hemos de añadir la clausula **defaults** en la ruta asociada:



Si no indicamos el parámetro nombre nos llega Nombre\_por\_defecto.

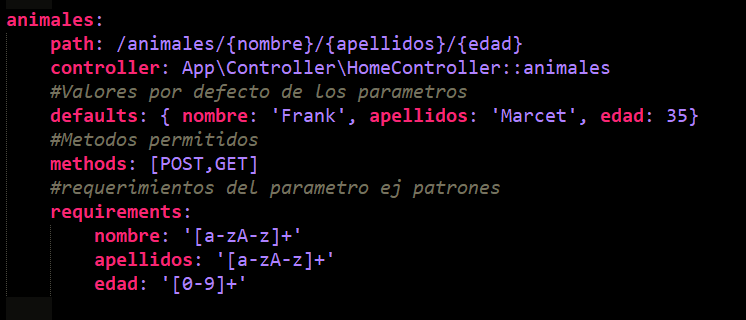
### 4.2.2 PARAMETRO OPCIONAL SIN VALOR POR DEFECTO.



Asi al controlador animales($nombre), recibe $nombre = null

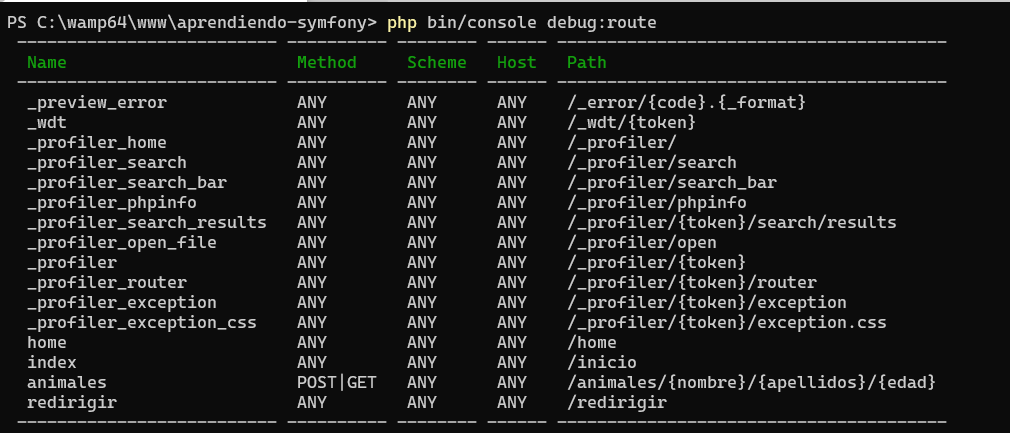
## 4.3 RUTAS AVANZADAS.

Vamos a añadir mas clausulas



## 4.4 MOSTRAR RUTAS POR CONSOLA.

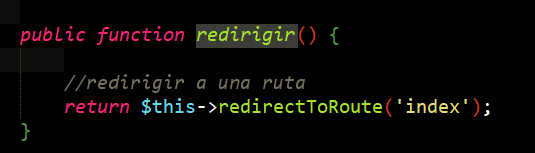
Vamos a usar el comado:



## 4.5 REDIRECCIONES.

Podemos redireccionar a otras paginas de diferentes formas. Vamos a crear un nuevo controlador para ver estas formas:

### 4.5.1 REDIRIGIR A UNA RUTA



### 4.5.2 REDIRECCION 301.

Las redirecciones 301 son bastante simples. Se utilizan para redirigir una página web a otra.

**¿Qué es una redirección 301?**

Una redirección 301 indica el cambio permanente de una página web de una ubicación a otra.

La parte 301 se refiere al código de estado HTTP de la página redirigida.

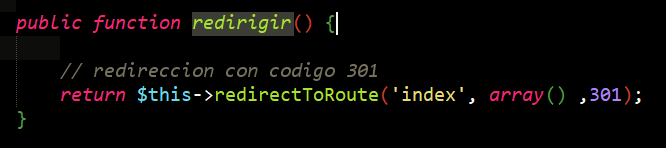
**Ejemplo:**[*blog.ahrefs.com*](http://blog.ahrefs.com/) redirige a [*ahrefs.com/blog*](https://ahrefs.com/blog)

En términos simples, una redirección 301 le dice al navegador: “Esta página se ha movido permanentemente. Esta es la nueva ubicación y no tenemos la intención de regresar a la anterior”. A lo que el navegador responde: “¡Claro que sí! ¡Enviaré al usuario allí ahora mismo!”

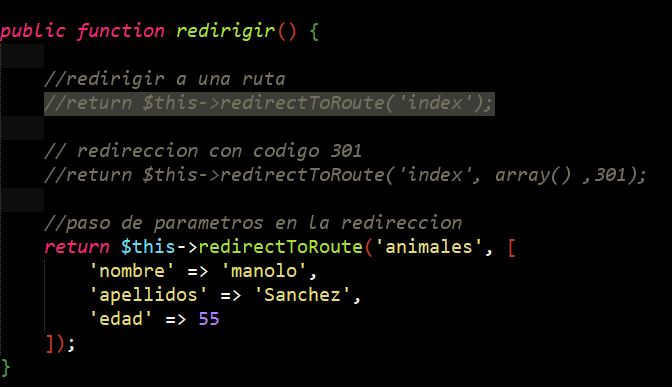
Es por eso que si intentas visitar el *blog.ahrefs.com*, no podrá suceder.

En su lugar terminarás en *ahrefs.com/blog*.

Es optimo para el SEO



### 4.5.3 REDIRECCION CON PASO DE PARAMETROS.



### 4.5.4 REDIRECCION A UNA URL.



# 5. VISTAS Y MOTOR DE PLANTILLAS TWIG.

## 5.1 PLANTILLA BASE.

Vamos a crear una plantilla base:

<!DOCTYPE html>

<html lang="es">

    <head>

        <meta charset="UTF-8">

        <meta name="viewport" content="width=device- width, initial-scale=1.0">

        <!-- Estilos -->

        {% block stylesheets %}{% endblock %}

        <!-- Js -->

        {% block javascripts %}{% endblock %}

        <title>

            {% block titulo %}

                Titulo de la plantilla

            {% endblock %}

        </title>

    </head>

    <body>

        <!-- Header -->

        <div id="header">

            {% block header %}

                <header id="header">

                    <h1>Cabecera de la plantilla</h1>

                </header>

            {% endblock %}

            {% block nav %}

                <nav>

                    <a href="">Inicio</a>

                    <a href="">IR SECCIÓN 2</a>

                    <a href="">IR SECCIÓN 3</a>

                </nav>

            {% endblock %}

        </div>

        <!-- End Header -->

        <!-- Section -->

        <section id="section">

            {% block name %}

                <p>contenido de la plantilla</p>

            {% endblock %}

        </section>

        <!-- End Section -->

        <!-- Footer -->

        <footer id="footer">

            {% block footer %}

                <p>Pie de pagina de la plantilla</p>

            {% endblock %}

        </footer>

        <!-- End Footer -->

    </body>

</html>

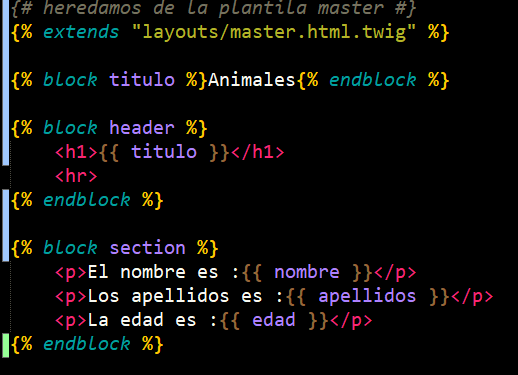
## 5.2 HEREDAR PLANTILLA BASE EN UNA VISTA.

Vamos a heredar de la plantilla master que acabamos de crear en animales.html.twig

NOTA

Si no ponemos el bloque en la plantilla de la cual se hereda automáticamente se muestra lo de la plantilla.

En este ejemplo el header lo modificamos nosotros asi que no se ve el texto de la plantilla.



Y este seria el resultado

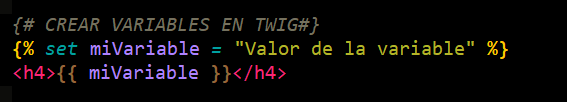


## 5.3 HEREDAR CONTENIDO DE LA PLANTILLA MASTER

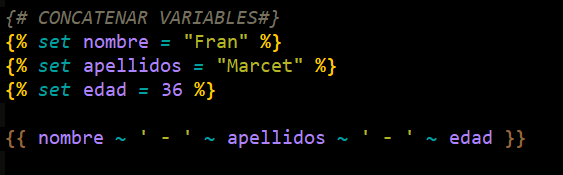
Para heredar el contenido de la plantilla y mostrarlo también en la otra plantilla usamos **{{ parent }}**



## 5.4 CREAR VARIABLES EN TWIG.



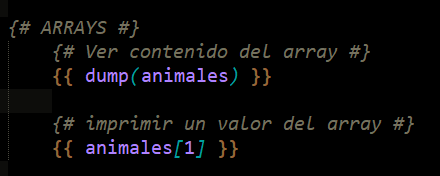
## 5.5 CONCATENAR VARIABLES.



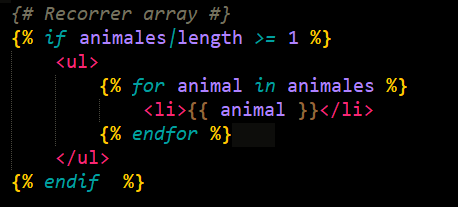
## 5.6 ARRAYS EN TWIG.

### 5.6.1 ARRAYS

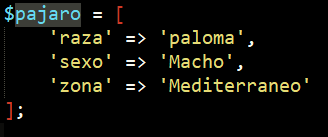
Vamos a devolver desde el controlador un array -> animales y veamos como se trabaja con el

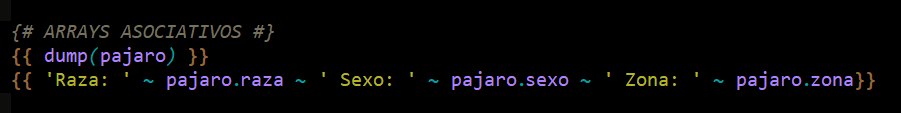


#### 5.6.1.1 RECORRER ARRAY



### 5.6.2 ARRAYS ASOCIATIVOS

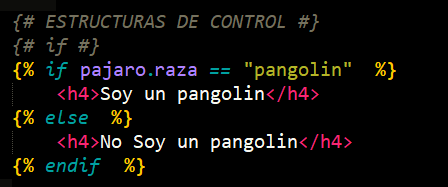


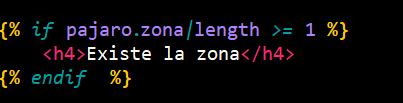


RECORRER ARRAY

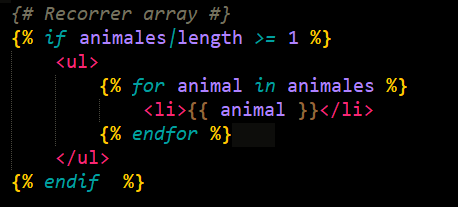
## 5.7 ESTRUCTURAS DEL CONTROL.

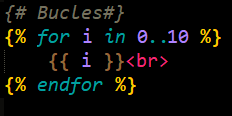
### 5.7.1 IF – ELSE.

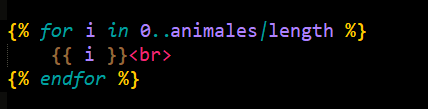




### 5.7.2 BUCLES.





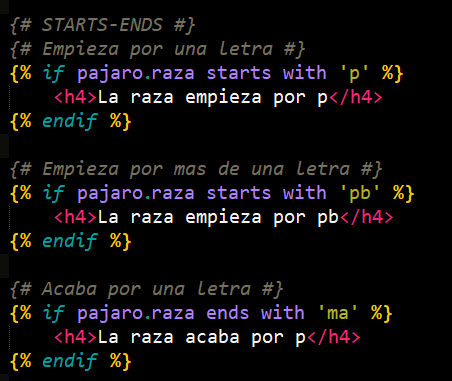


<https://twig.symfony.com/doc/3.x/tags/for.html>

### 5.7.3 STARTS-END.

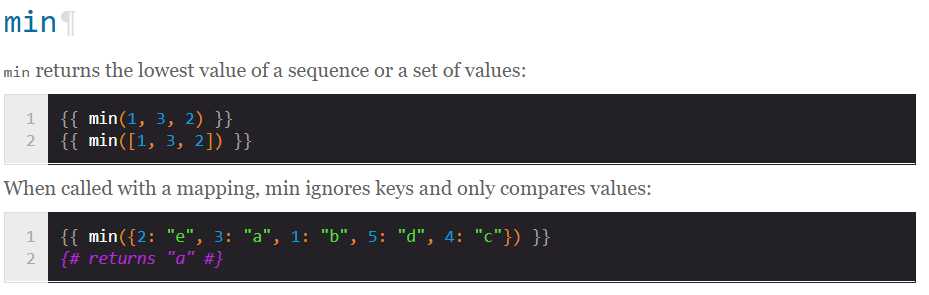
Imaginemos que deseamos evaluar si del array pajaros :

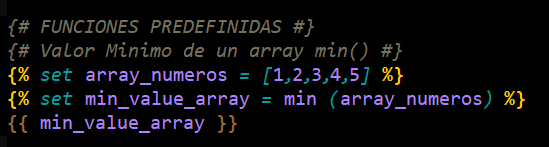




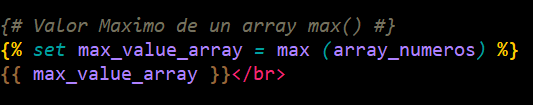
## 5.8 FUNCIONES PREDEFINIDAS EN TWIG.

### 5.8.1 VALOR MINIMO Min().

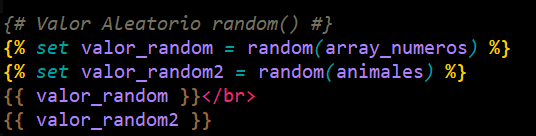




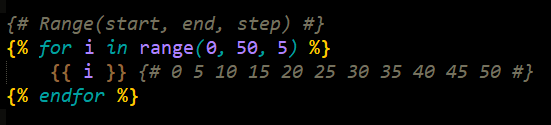
### 5.8.2 VALOR MAXIMO Max().



### 5.8.3 ELEMENTO ALEATORIO DE UN ARRAY.



### 5.8.4 RANGE().



## 5.9 INCLUDES.

### 5.9.1 INCLUDE BASICO.

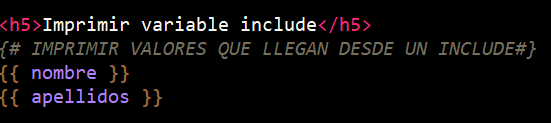


### 5.9.2 INCLUDE CON PASO DE PARAMETROS.

Por ejemplo vamos a pasarle un Json



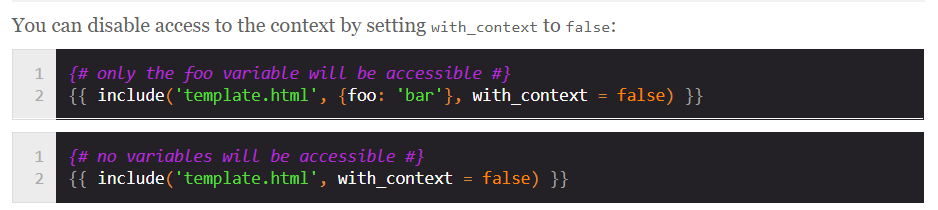
Y ya estaría disponibles en la vista funciones.html.twig:



### 5.9.3 CLAUSULA with\_context = false

Podemos deshabilitar el uso de variables mediante la clausula, pudiendo solo acceder a las variables indicadas

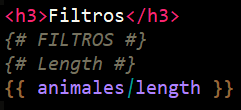




## 5.10 FILTROS.

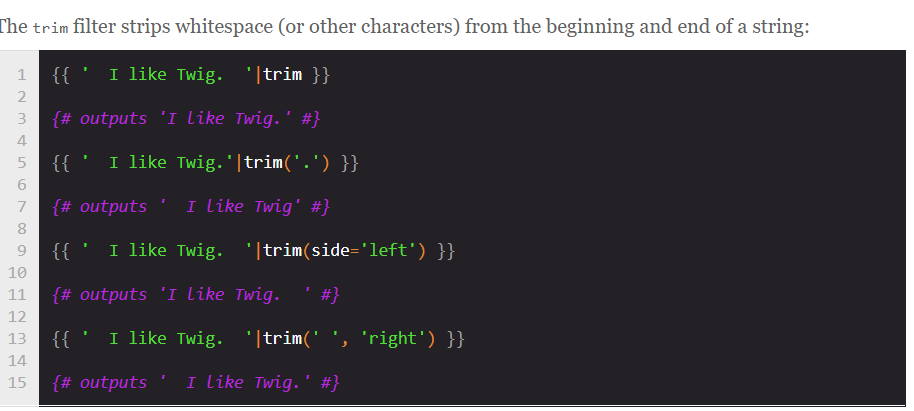
Son pipes o tuberías que nos permiten modificar variables, objetos.

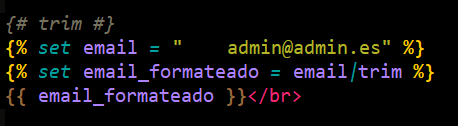
### 5.10.1 LENGTH



### 5.10.2 TRIM.

Formatear espacion o caracteres





### 5.10.3 ENCADENAR FILTROS.

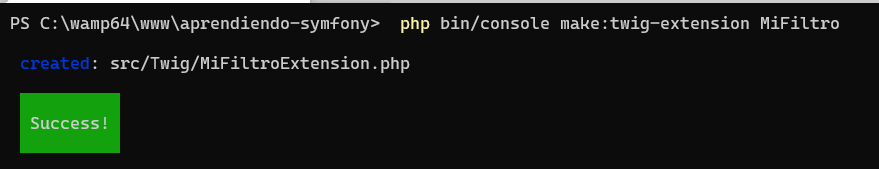


## 5.11 CREAR EXTENSIONES.

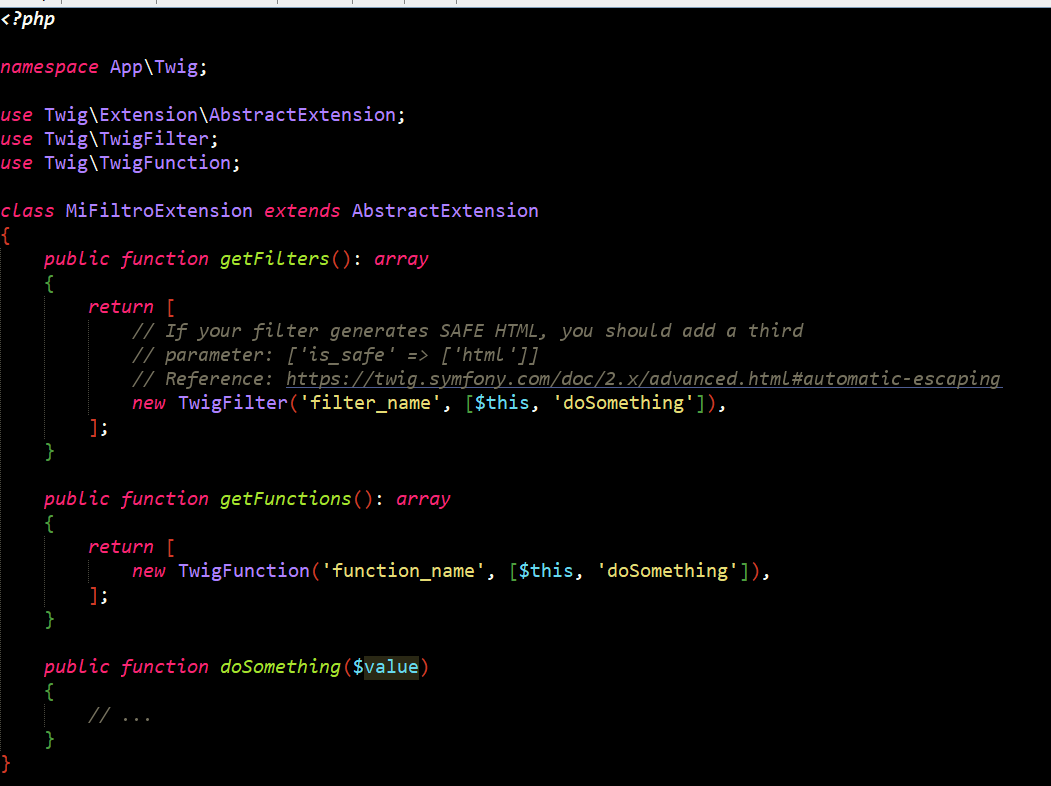
Vamos a crear una extensión que sea un filtro y posteriormente usarla en cualquier vista de twig.

Creamos en src la carpeta Twig -> src/Twig/Filtro.php.

Podemos usar el comando:



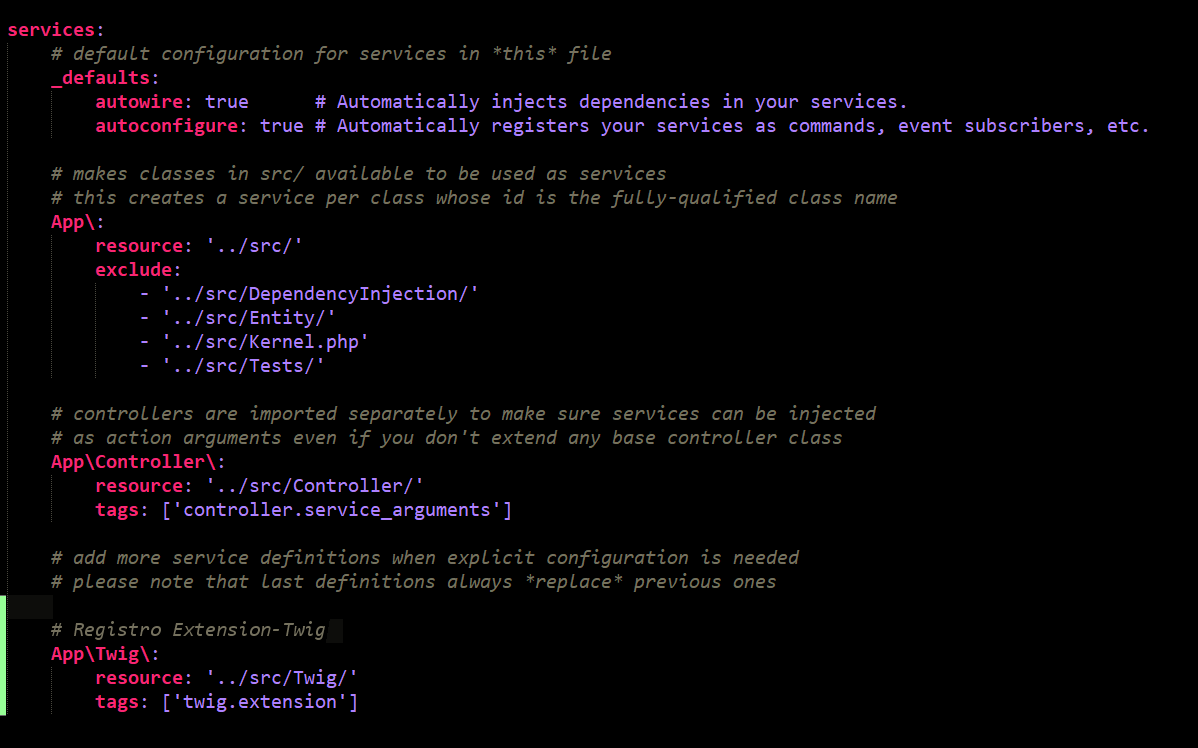
Al crear el filtro tenego algo como esto:



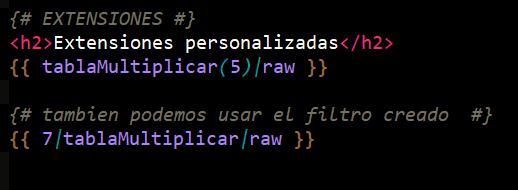
Vamos por ejemplo a hacer una tabla de multiplicar que sirva de extensión y de filtro:



Una vez creada tenemos que registrar el servicio en symfony en config/services:



Vamos a usar la extensión en la vista, usamos el filtro raw:



# 6. BASES DE DATOS EN SYMFONY.

## 6.1 CONFIGURACION DE LA BD.

Vamos al fichero .env, y modificamos:

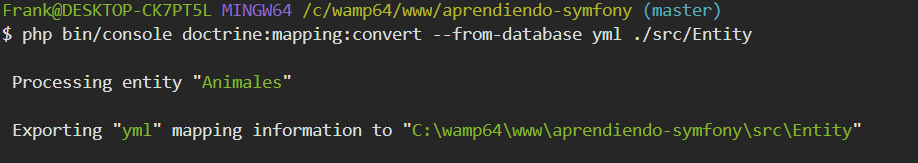


Para la creación de la BD, podemos hacerlo de manera normal en phpmyadmin o usando el comando:

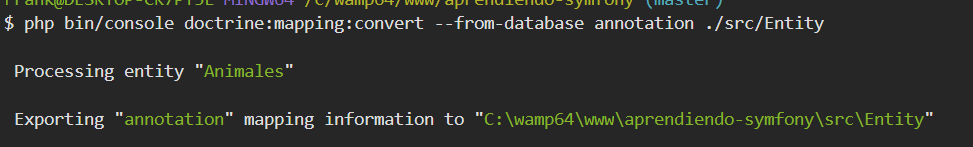


## 6.2 CREACION DE ENTIDADES DESDE LA BASE DE DATOS.

Vamos a crear la tabla animales, mediante SQL y posteriormente pasarla a entidad de doctrine , vamos a utilizar el formato yml, mediente la clausula **yml** para obtener un fichero yml con los tipos de datos y emas de las variables:



Y mediante la clausula **annotation**:



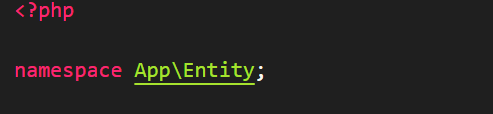
**CREACION GETTERS Y SETTERS.**

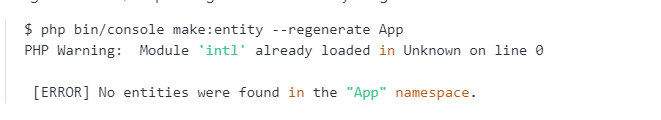
Vamos a usar el comando



NOTA

Si no añadimos, no dara EROR ya que no encuentra el namesapce





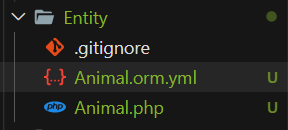
Por ultimo volvemos a usar el comando. Una vez obtenido el fichero con las clases para trabajar con formato yml:



CAMBIAR NOMBRES A SINGULAR.

Vamos a cambiar el nombre a singular ya que de la clase tanto en el fichero





## 6.3 GENERAR ENTIDADES DESDE SYMPONY.

Ahora vamos a hacerlo al revés vamos a generar la entidad y luego crear la tabla en la BD. Para ello usamos el comando:



Podemos añadir campos mediante la consola fácilmente.

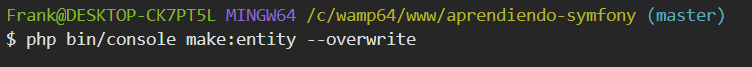
En caso de tener que modificar el fichero de entidad Usuario haremos los cambios y ejecutamos el comando:



Nos crea los getter y setters que nos falten en caso de no estar implementados.

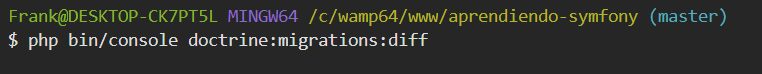
### 6.3.1 SOBRESCRIBIR ENTIDAD(Añadir registros).

De nuevo no deja añadir campos mediante la consola y genera automáticamente los getters y setters.

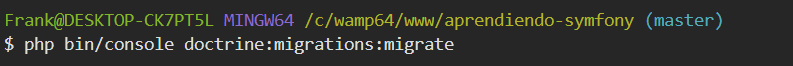


## 6.4 GENERACION DE TABLAS A PARTIR DE ENTIDADES.

Una vez definidas las entidades vamos a generar las tablas mediante las migraciones de Symfony :

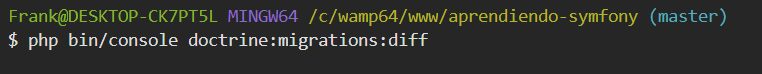


Ahora vamos a ejecutar la migración para que se efectúen los cambios en la BD

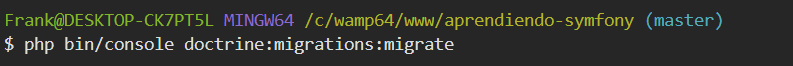


## 6.5 MODIFICAR ENTIDADES.

Podemos efectuar los cambios y ejecutar como antes



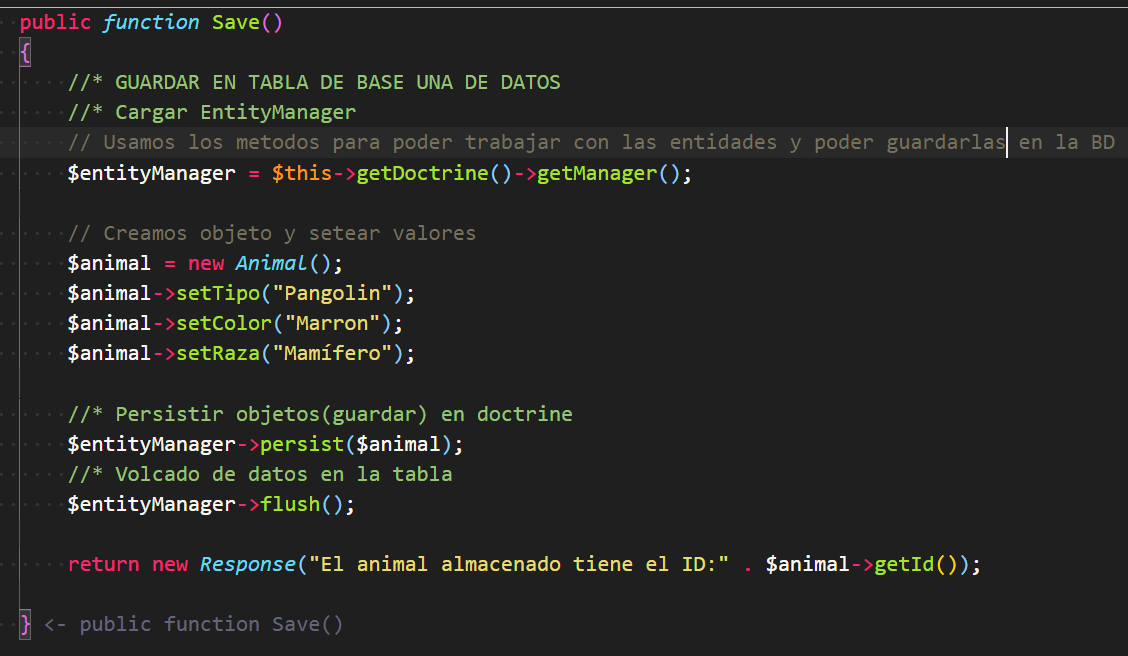
Y posteriormente :



Y ya tendríamos los campos

## 6.6 GUARDAR EN LA BASE DE DATOS.

Vamos a generar un método para guardar los registros en la base de datos:

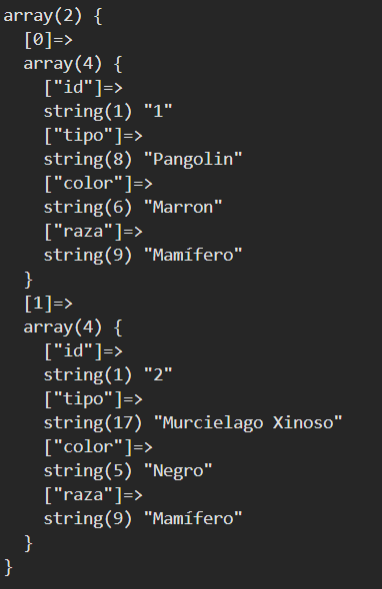


## 6.7 CONSULTAS EN LA CONSOLA DE DOCTRINE.

Podemos generar consultas en la consolas y asi poder ver el resultado de las consultas o comprobar que se realizo bien un insert.

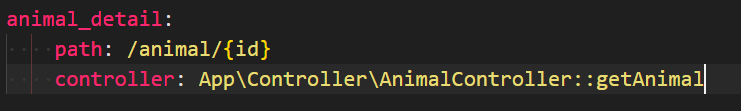


Y nos devuelve un result set en forma de array en la consola.

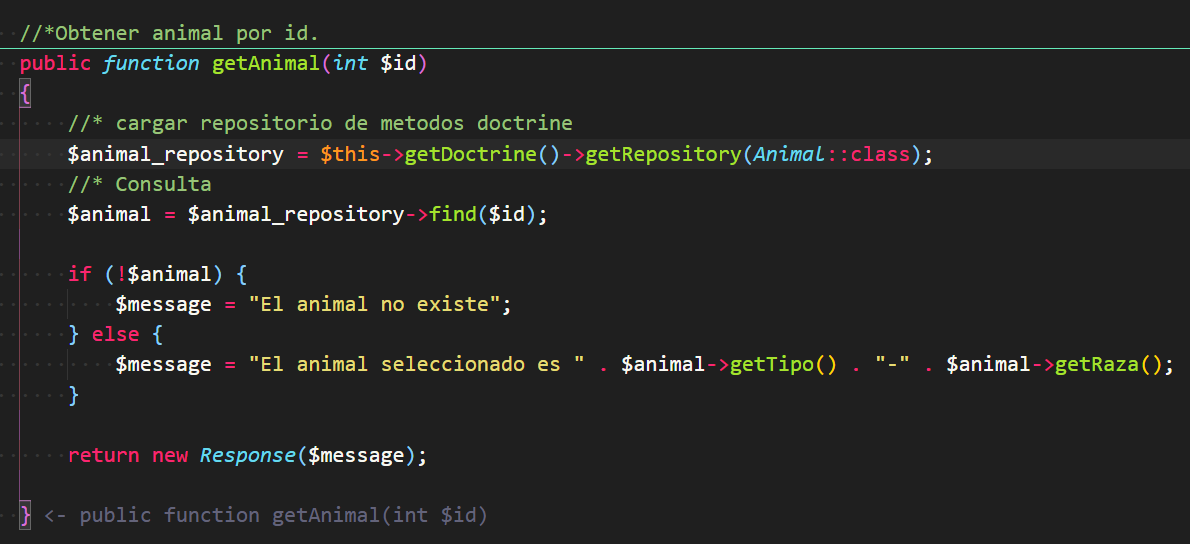


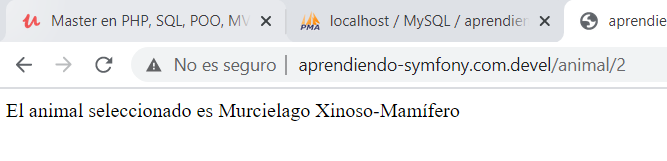
## 6.8 CONSULTAS ORM.OBTENER REGISTRO.find().

Vamos a crear una ruta para el ejemplo:



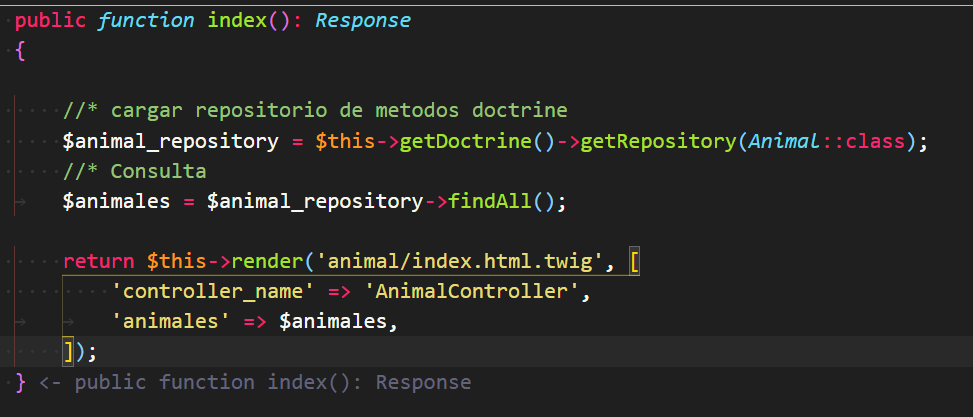
Metodo





## 6.9 CONSULTAS ORM.LISTADO DE REGISTROS.findAll().

Vamos a obtener todos los registros de una tabla. Es similar a find(), pero con el método findAll()

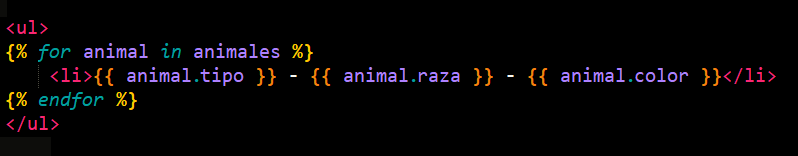


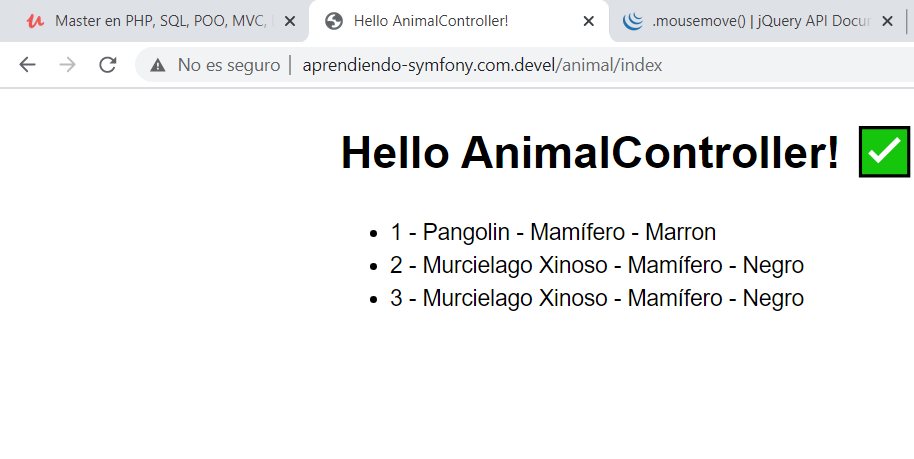
Ahora en la vista vamos a recorrer la variable $animales que sera un array con objetos en su interior.

Hagamos un dump(), en la vista:



Simplemente con un for in



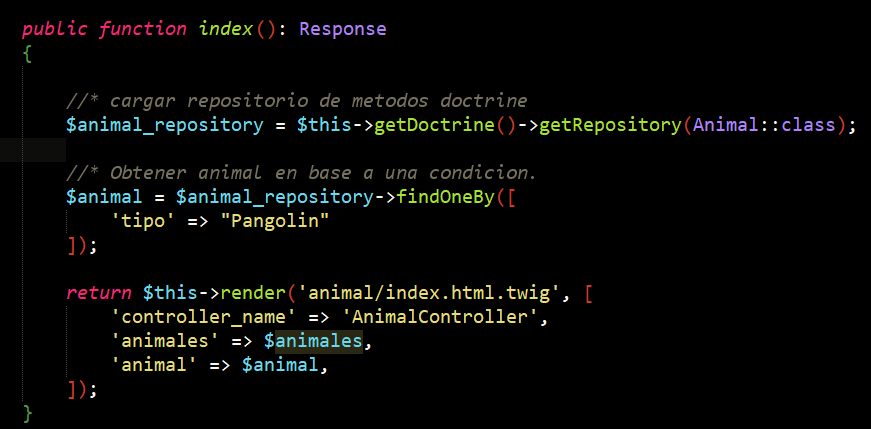


## 6.10 CONSULTAS ORM.CONSULTAS.TIPOS DE find().

### 6.10.1 OBTENER REGISTRO EN BASE A UNA CONDICION findOneBy().

Utilizaremos el método findOneBy(), y le pasaremos como parámetros un array con las condiciones:

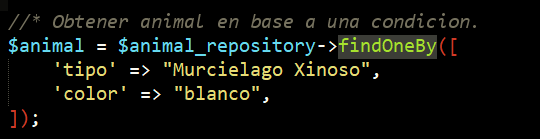
Es similar a un **SELECT \* FROM animales WHERE tipo =”Pangolin”**



NOTA

Podemos añadir mas de una condición al array:

Es similar a un **SELECT \* FROM animales WHERE tipo = ”Pangolin”** **AND color = “blanco”**



### 6.10.2 CONSULTAS ORM.OBTENER VARIOS REGISTROS EN BASE A UNA CONDICION findBy().

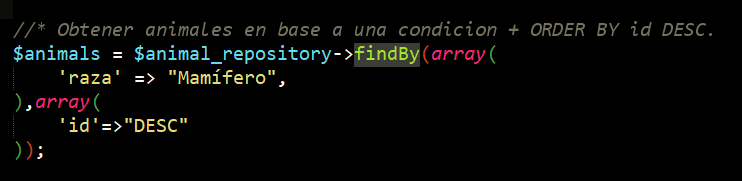
Tenemos también el método para obtener una lista de registros en base a una condición.



### 6.10.3 CONSULTAS ORM.AÑADIR CLAUSULA ORDER BY.

Hemos de añadir un segundo array a la condición:

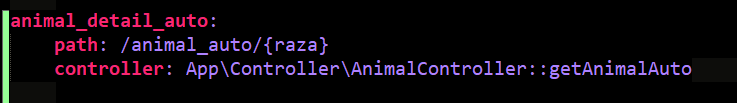
**SELECT \* FROM animales WHERE raza = “Mamifero” ORDER BY id DESC**



## 6.11 CONSULTA ORM.OBTENER OBJETO/REGISTRO AUTOMATICAMENTE.

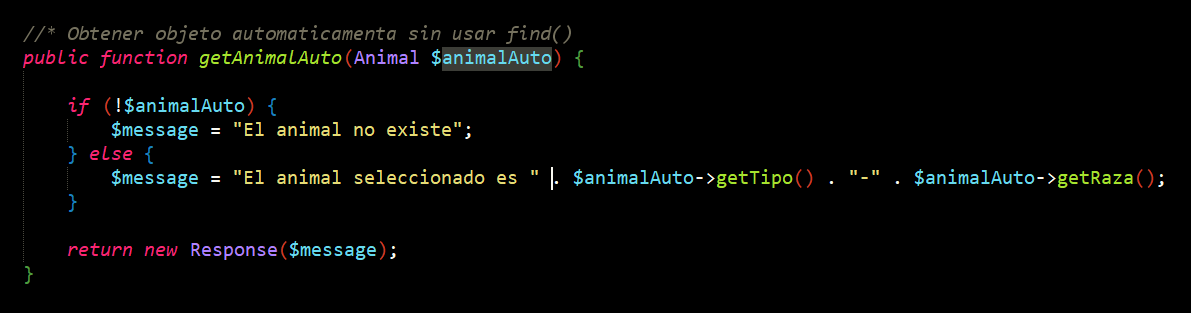
Podemos obtener un objeto de manera automática sin ejecutar los métodos find() anteriormente mencionados.

Vamos a crear una nueva ruta:

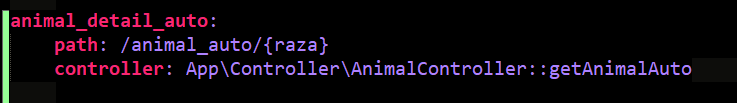


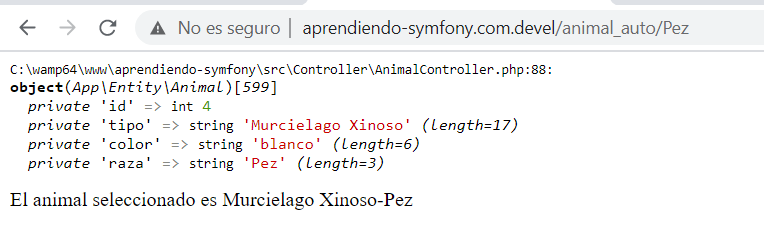
Y además de un nuevo método en el controlador:

Le pasaremos como parametro al método el objeto (Clase) del cual queremos obtener la información en este caso Animal

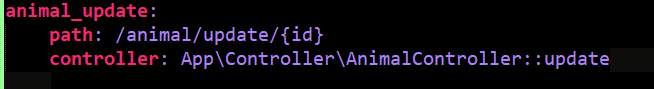


Veamos al insertar los valores por URL como obtenemos el objeto con la raza deseada, ya que es el parámetro que recibe por GET.



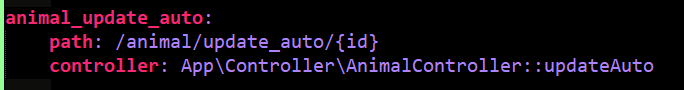


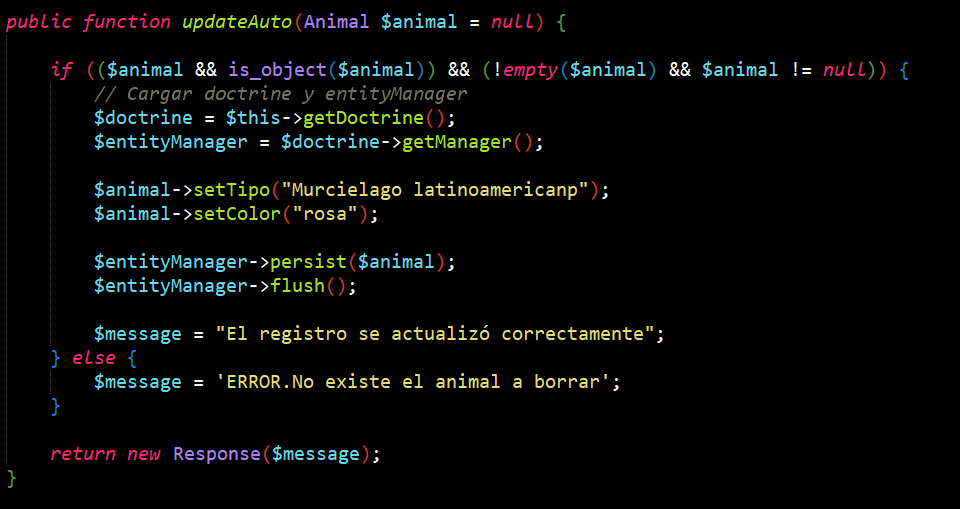
## 6.12 CONSULTAS ORM. ACTUALIZACION DE REGISTROS EN LA BD.





Tambien se podría realizar mediante el método automatico sin utilizar find()





## 6.13 ELIMINAR DE REGISTROS EN LA BD.

Metodo auto para buscar el elemento.



## 6.14 QUERY BUILDER.

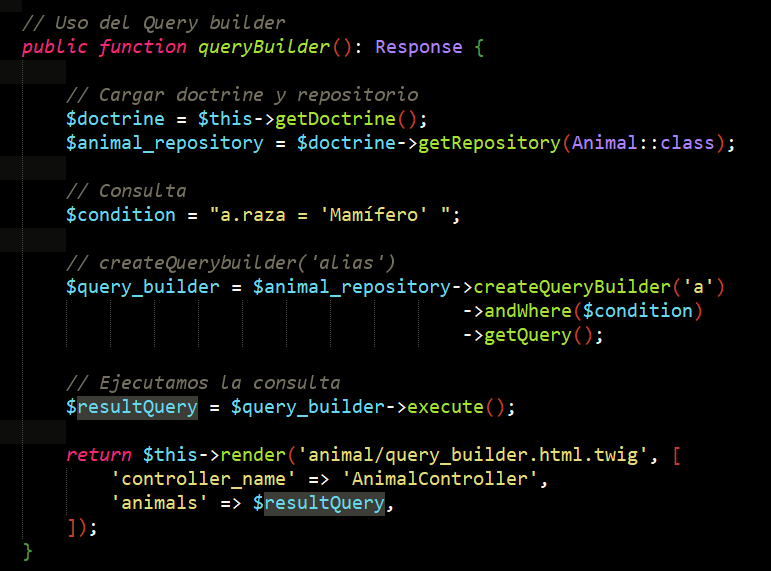
https://www.doctrine-project.org/projects/doctrine-orm/en/2.7/reference/query-builder.html

### 6.14.1 CONDICIONES.

Aparte de del método find y derivados para obtener registros tenemos un query builder en symfony. Las condiciones las implementamos mediante las clausula:

* andWhere()
* setParameter()

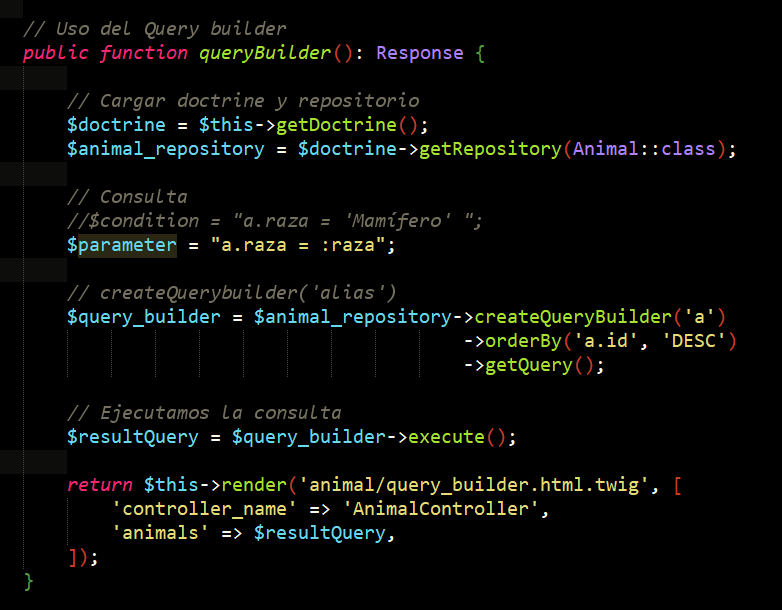
**Ejemplo andWhere()**



**Ejemplo setParameter**



### 6.14.2 ORDER BY.



## 6.15 LENGUAJE DQL.

Es un lenguaje de doctrine parecido a SQL pero haciendo referencia a los objetos y entidades.



## 6.16 LENGUAJE SQL.

Tambien podemos usar SQL nativo



## 6.17 CREAR REPOSITORIOS.

Un repositorio es una clase en la cual vamos a tener una serie de métodos para ejecutar consultas cuando estas tengan cierta complejidad, y de esta manera tendremos el controlador con el codigo mas legible.

si no hemos se nos ha creado al generar la entidad podemos crear un fichero en src/Repository.

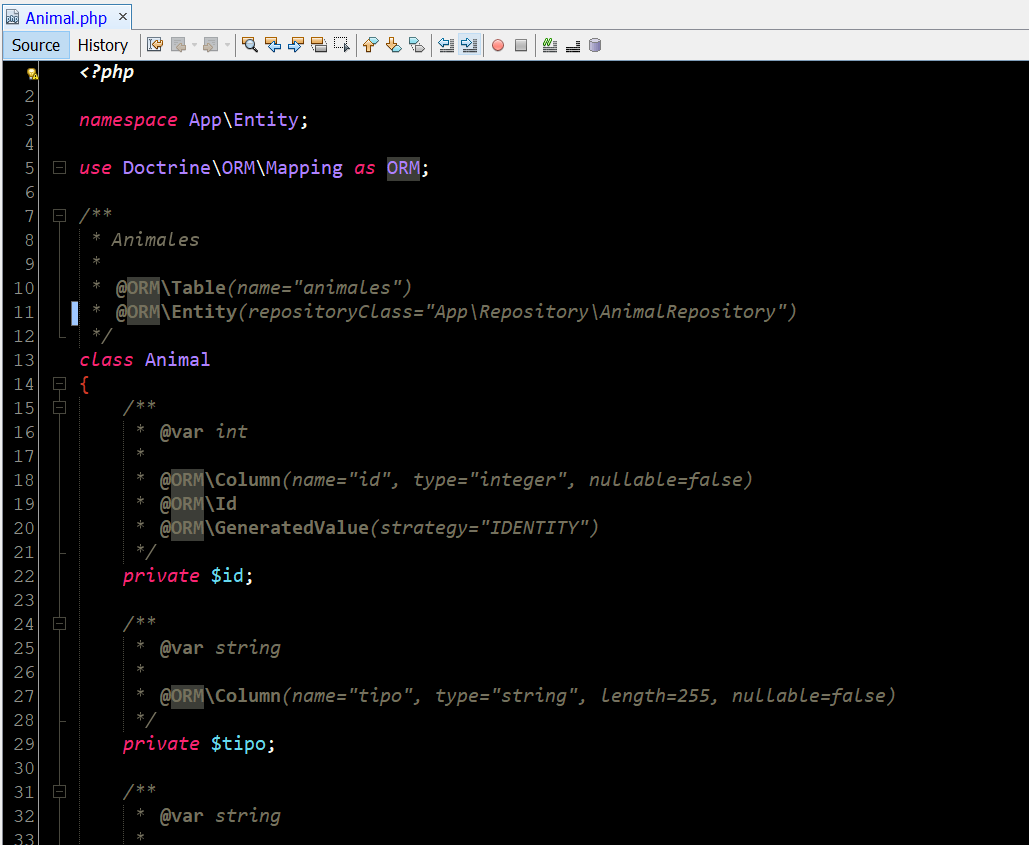
En primer lugar hemos de indicar:

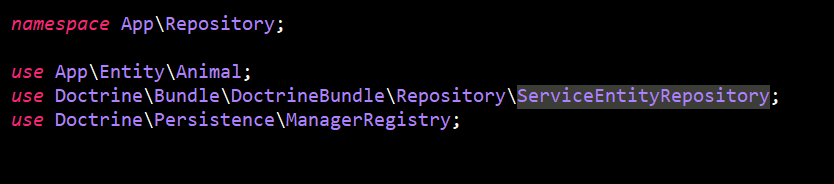
* namespace
* clase
* repositorios de doctrine en symfony

NOTA

MUY IMPORTANTE hemos de vincular nuestra clase o entidad con el repositorio sino nos devolvera siempre resultados empty.

Para ello vamos a la clase





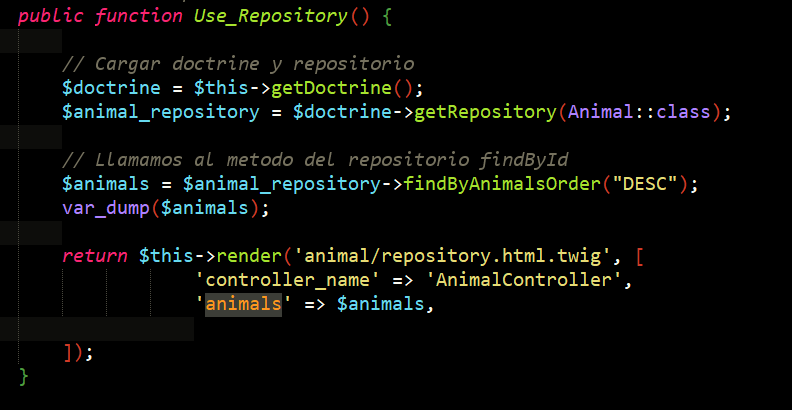
Ahora ya podemos crear la clase, el constructor y los métodos para ejecutar consultas:

AnimalRepository.php



Para usar el metodo creado en el repositorio, lo implementamos en el controlador

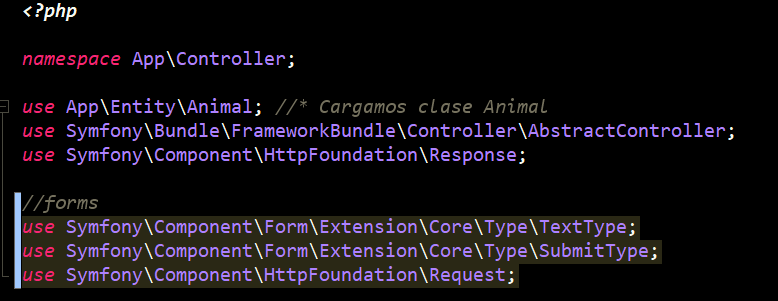
Metodo del controlador:



# 7. FORMULARIOS.

## 7.1 CREAR FORMULARIOS.

En primer lugar vamos a importar las librerías necesarias para trabajar con formularios en symfony:

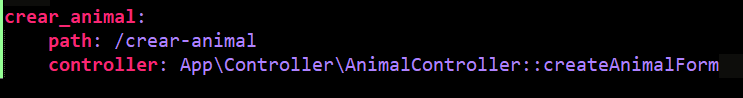


Vamos a crear un método para generar el formulario **createAnimalForm()** y renderizarlo en una vista

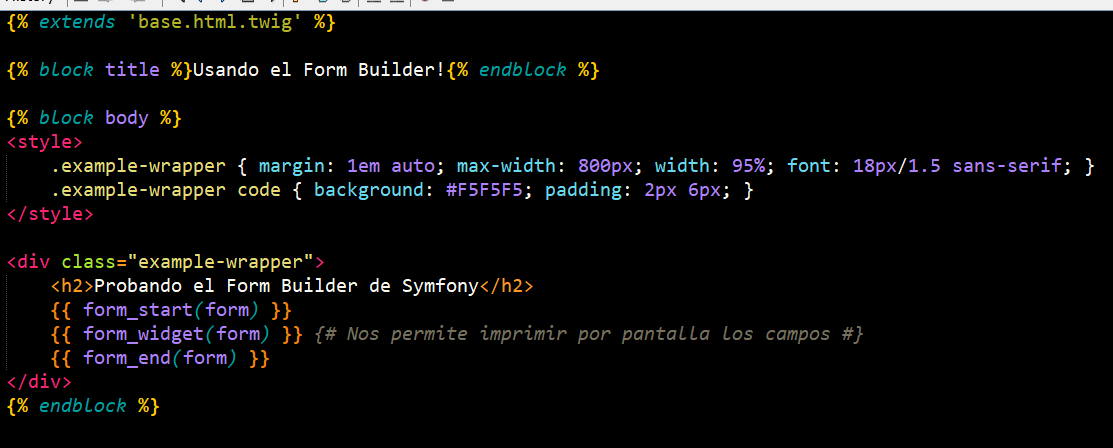
**createAnimalForm()**



**Ruta**



**Vista**



## 7.2 PERSONALIZAR CAMPOS.

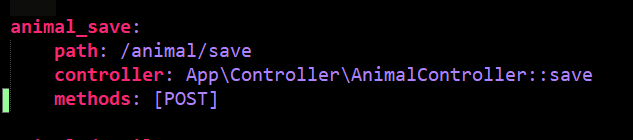
A cualquiera de los campos generados, le podemos añadir un 3º parámetro, un array con mas atributos.

Por ejemplo en el campo submit podemos cambiar el value:



## 7.3 RECIBIR DATOS DEL FORMULARIO.

El formulario llama a la siguiente ruta, gestionada por el controlador:



Pero realmente no es necesario, asi que podemos prescindir de la clausula :



Podemos gestionar eso desde el mismo método donde generamos el formulario -> **createAnimalForm()**

public *function* createAnimalForm(*Request* $request)

  {

      //\* Creamos nuevo obj

      $animal = new *Animal*();

      //\* Generamos el formulario

      $form = $this->createFormBuilder($animal)

      //->setAction($this->generateUrl("animal\_save"))

          ->setMethod("POST") //Por defecto es POST

          ->add("tipo", *TextType*::class, [

                'label' => 'Tipo de animal',

                //'help' => 'Inserte un tipo de animal',

            ])

          ->add("raza", *TextType*::class)

          ->add("color", *TextType*::class)

          ->add("submit", *SubmitType*::class, [

                'label' => 'Enviar', // Value

                'attr' => ['class' => 'btn'],

            ])

            ->getForm();

      //\* Recoger datos del formulario y los almacena

      //\*  en el obj en este caso $animal

      $form->handleRequest($request);

      // Comprobamos si se ha enviado el form

      if ( $form->isSubmitted() && $form->isValid() ) {

          //var\_dump($animal); die();

          //\* Almacenar datos en la BD

          $doctrine = $this->getDoctrine();

          $entityManager = $doctrine->getManager();

          $entityManager->persist($animal);

          $entityManager->flush();

          //\* session flash

          $session = new *Session*();

          //$session->start(); // No es necesario

          $session->getFlashBag()

->add('message', 'Animal creado correctamente');

            return $this->redirectToRoute("crear\_animal");

       }

        //\* Renderizamos la vista

        return $this->render("animal/add\_animal.html.twig", [

            'form' => $form->createView(),

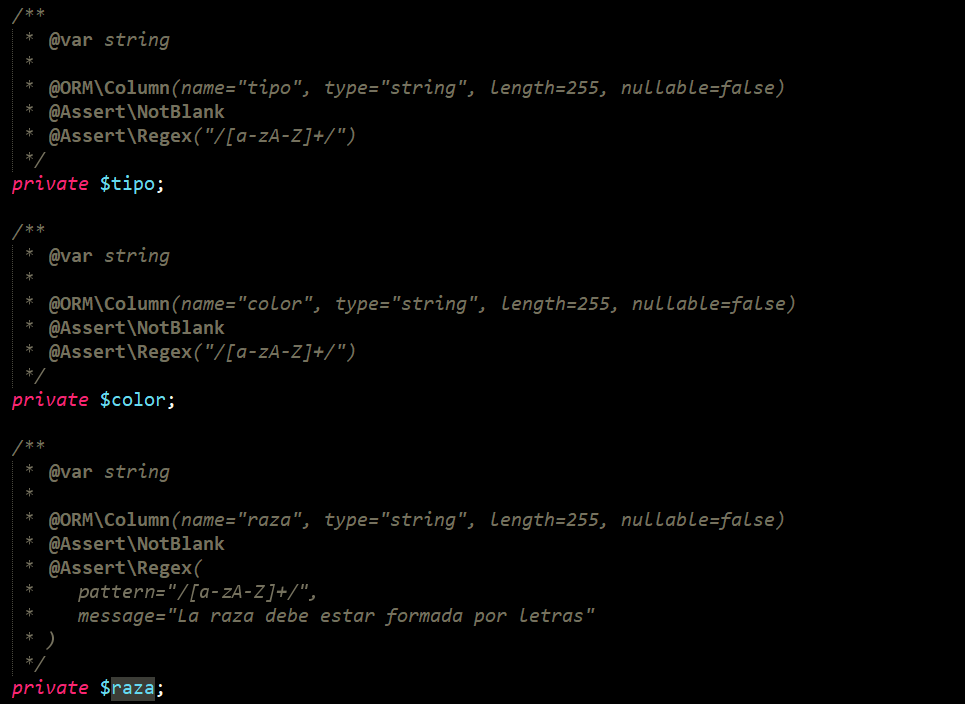
        ]);

    }// END CLASS

## 7.4 VALIDACION DE CAMPOS.

Añadimos la librería de validaciones en la entidad





## 7.5 FORMULARIOS SEPARADOS EN CLASES.

Hasta ahora hemos definido el formulario dentro de un método del controlador. Pero es mucho mejor tenerlos separados en una o varias clases y asi poder reutilizarlos en el proyecto.

Creamos en Entity un directorio Form.

Se les suele nombrar como **Nombre\_entidadType.php**, por ejemplo AnimalType.php



Importamos en nuestro controlador la librería para podre acceder a la clase y poder usar el objeto.



Ahora en el controlador cambiamos el método que teníamos



Por



Quedando el controlador mucho mas limpio y mantenible.



## 7.6 VALIDACION DE DATOS AISLADOS.

Hasta ahora hemos validado los datos de un formulario generado por symfony, pero también podemos validar datos desde un formulario creado en html y enviando los datos al controlador.

En este ejemplo vamos a validar un email que nos llega por GET:

LIBRERÍA VALIDATOR

