

Aplicación con Symfony



Symfony

Cristian Leandro Sánchez Mego

2º DAW

Sumario

1. Acerca de Symfony:	3
1.1. Modelo Vista-Control:	3
2. Instalación:	4
2.1. Instalación del CLI:	4
2.2. Creación del proyecto:	5
3. Instalación de Bundles para la producción y el desarrollo:	5
3.1. Make Bundle:	5
3.2. Doctrine annotations:	5
4. Creación de la base de datos:	6
4.1. Estructura de la base de datos:	6
4.1.1. Modelo entidad-relación:	8
4.1.2. Entidades:	8
4.1.3. Relación de cada entidad:	8
4.2. Creación de Entidades:	9
4.2.1. Creación de relaciones:	9
4.3. Resultado en las bases de datos:	12
4.4. Creación de datos base para la base de datos:	13
4.4.1. Inserción de datos:	14
5. Creación de controladores, vistas y formularios con make:crud:	14
5.1. Paginación con KnpPaginator:	18
6. Consultas específicas:	20
7. Control de acceso:	22
8. Conclusiones:	24
9. Bibliografía:	25
10. Anexos:	25
10.1. Controladores:	25
10.1.1. Aviones:	25
10.1.2. Bases:	28
10.1.3. Personal:	30
10.1.4. Vuelos:	32
10.2. Repositorio:	35
10.2.1. Aviones:	35
10.2.2. Bases:	36
10.2.3. Personal:	37
10.2.4. Vuelos:	38

1. Acerca de Symfony:

Symfony es un Framework desarrollado en PHP con el objetivo de facilitar las tareas de desarrollo tanto en front-end cómo en back-end. Se apoya en Doctrine (encargado de la gestión de repositorios) y en bundles de terceros para así poder ofrecer una experiencia de uso y gestión tanto para la codificación, el manejo y la conexión a base de datos y el diseño de las páginas.

1.1. Modelo Vista-Control:

Symfony adapta el modelo vista control (MVC), el cuál se basa en tres ejes:

- **Modelo:** Es la representación de la información. Representa la lógica del negocio y maneja la base de datos de la aplicación (exactamente el ORM de Doctrine, que con el enrutamiento y los datos obtenidos del cliente, traduce las tablas en objetos y cada columna cómo una propiedad del objeto que crea).
- **Vista:** Es la parte front del proyecto, encargada de mostrar los resultados de la consulta o la operación solicitada por el cliente y trasladada por el modelo. Es la parte visible del proyecto que contiene formularios, plantillas visuales, imágenes, etc.
- **Controlador:** Encargo de recibir las consultas producidas por el cliente y transferidas en acciones por el modelo. Actualiza, muestra, crea y elimina la información que se le ordena mediante funciones y repositorios que se encargan de hacer peticiones a la base de datos y los elementos (tuplas) obtenidos se convierten en objetos que se pasan hacía la vista para que los muestre de manera ordenada (cómo en listas o tablas).

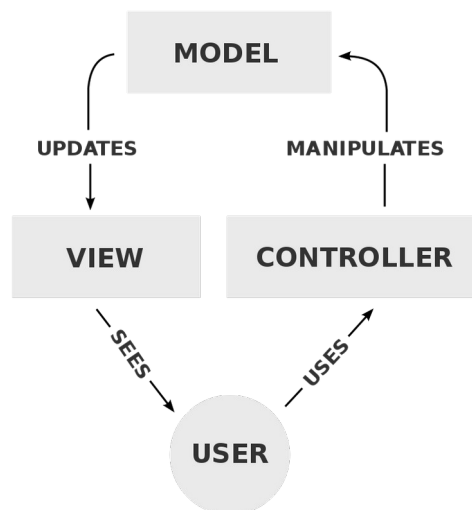


Figura 1: MVC

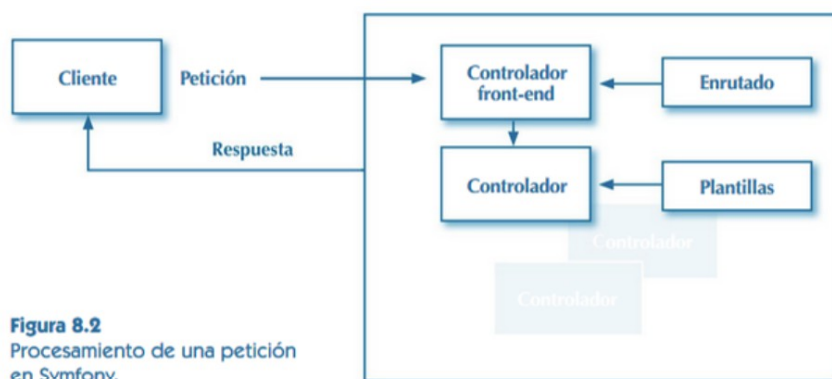


Figura 8.2
Procesamiento de una petición
en Symfony.

Figura 2: MVC en Symfony

2. Instalación:

Antes de empezar con el Symfony, necesitamos instalar el Composer y el Doctrine, luego de esto ya podemos proceder a instalar de manera global o en un directorio específico Symfony.

2.1. Instalación del CLI

Con esto instalaremos el binario `symfony`, que en el futuro nos ayudará también con los comandos básicos para iniciar el servidor.

```
wget https://get.symfony.com/cli/installer -O - | bash
```

Con esto podremos hacer la confirmación de los requerimientos, comprobando si nos faltara alguno:

```
symfony check:requirements
```

2.2. Creación del proyecto

Si tenemos todo en orden, ya estaremos habilitados para crear nuestro primer proyecto con el siguiente comando.

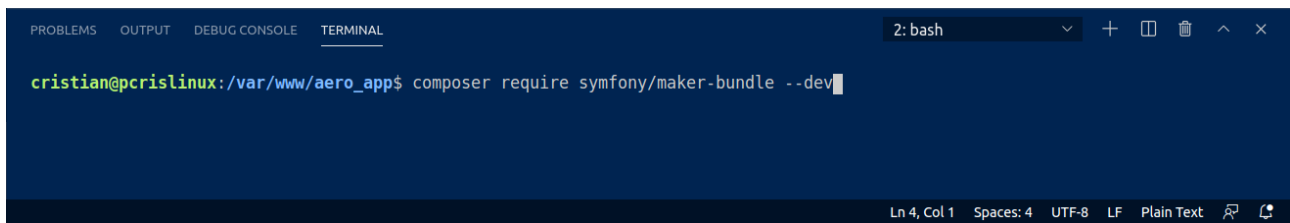
```
symfony new aerovuelo_app -full
```

Con esto tendremos todas las carpetas ya creadas listas para poder empezar el proyecto. Es importante tener previamente la carpeta contenedora del proyecto conectada al Git-Hub, ya que nos ayudará con las versiones de nuestro proyecto y tenerlo respaldo frente a cualquier eventualidad.

3. Instalación de Bundles para la producción y el desarrollo:

3.1. Make Bundle

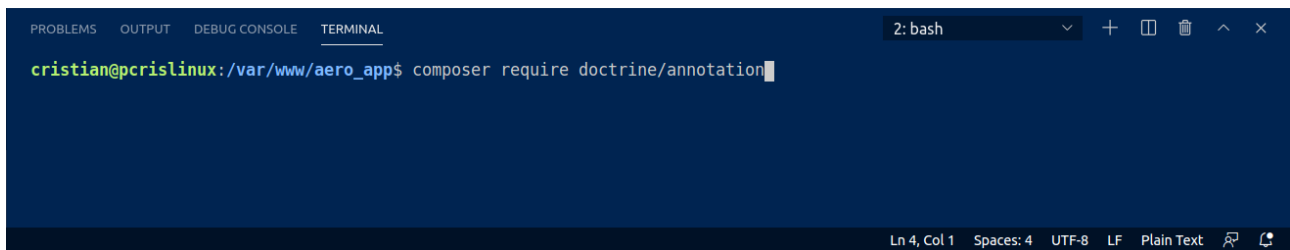
Este bundle es propio de Symfony, el cual nos ayudará para crear entidades, relaciones, usuarios, formularios, logins, etc. Es de los bundles más útiles que existen y nos ahorrarán muchas horas de código, cómo veremos adelante.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: bash
cristian@pcrislinux:/var/www/aero_app$ composer require symfony/maker-bundle --dev
Ln 4, Col 1 Spaces: 4 UTF-8 LF Plain Text
```

3.2. Doctrine annotations:

Necesario para los apuntes de Doctrine cuándo genera las entidades y los posibles errores que pueden haber en las relaciones, las consultas o las entidades disponibles.

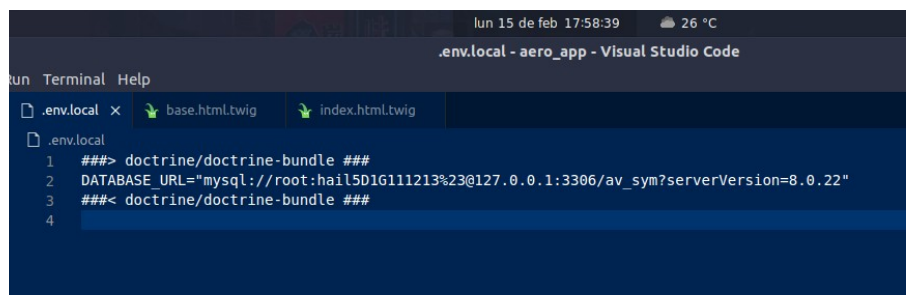


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: bash
cristian@pcrislinux:/var/www/aero_app$ composer require doctrine/annotation
Ln 4, Col 1 Spaces: 4 UTF-8 LF Plain Text
```

4. Creación de la base de datos:

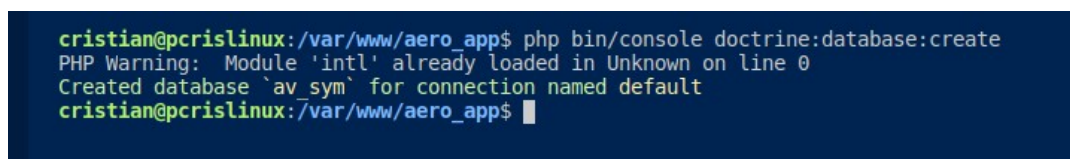
Para la base de datos he usado la combinación de MySQL, phpMyAdmin y Apache. Se crea de la siguiente manera:

- Primero, el archivo **.env** se debe copiar dentro de la misma carpeta del proyecto y llamarlo **.env.local**.
- Paso seguido se debe introducir los datos de conexión hacia la base datos de la siguiente manera:



```
lun 15 de feb 17:58:39 26 °C
.env.local - aero_app - Visual Studio Code
Run Terminal Help
.env.local x base.html.twig index.html.twig
.env.local
1 ###> doctrine/doctrine-bundle ###
2 DATABASE_URL="mysql://root:hail5D1G111213%23@127.0.0.1:3306/av_sym?serverVersion=8.0.22"
3 ###< doctrine/doctrine-bundle ###
4
```

- Por último, con el comando **php bin/console doctrine:database:create** podremos crear la base de datos en el MySQL.

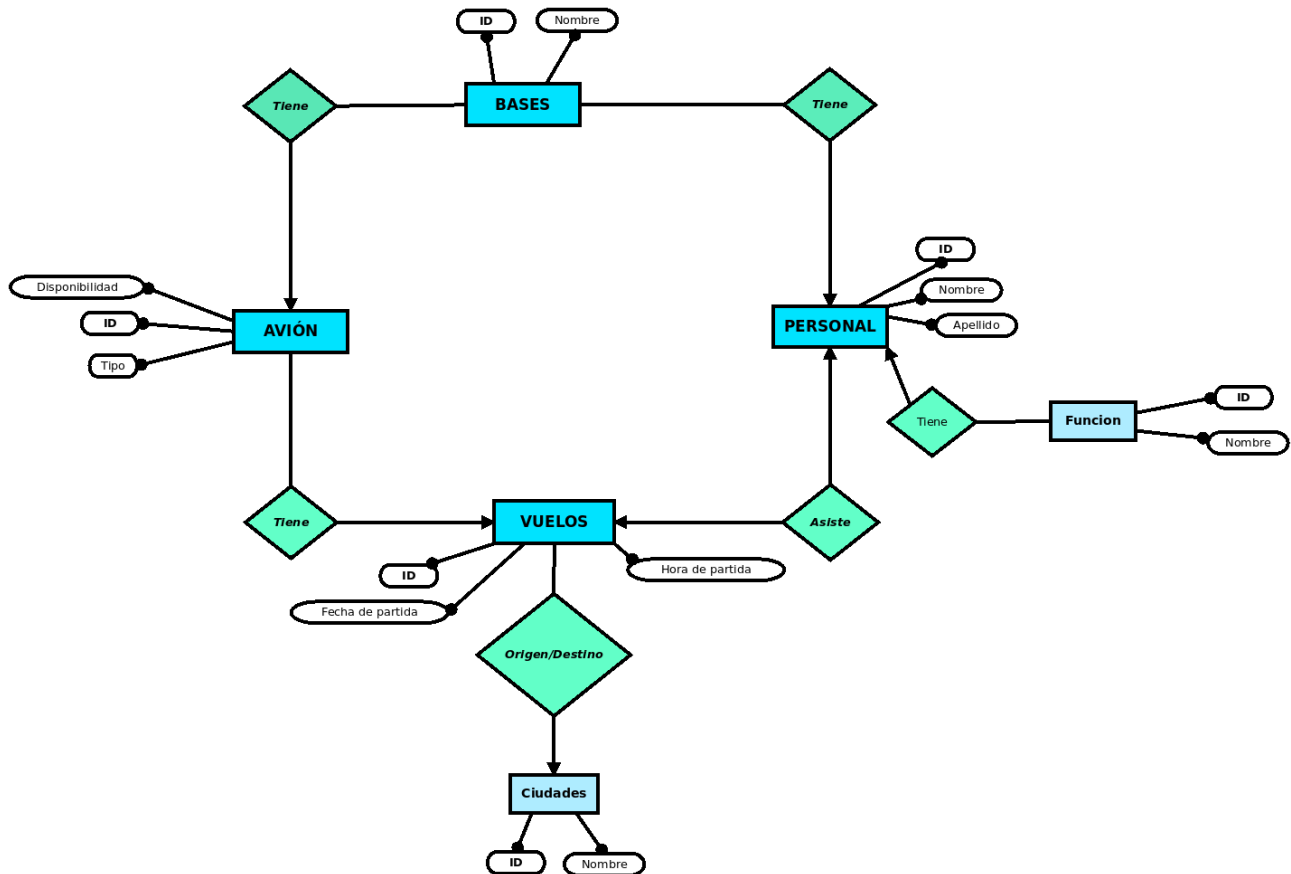


```
cristian@pcrislinux:/var/www/aero_app$ php bin/console doctrine:database:create
PHP Warning: Module 'intl' already loaded in Unknown on line 0
Created database 'av_sym' for connection named default
cristian@pcrislinux:/var/www/aero_app$
```

4.1. Estructura de la base de datos

Antes de crear las entidades y los controladores, es necesario mostrar los cambios que ha sufrido la base de datos con respecto al anterior proyecto.

4.1.1. Modelo entidad-relación



4.1.2. Entidades:

- Avión: Entidad que carga con los aviones .
- Personal: Personas que pertenecen a la compañía para efectuar vuelos.
- Vuelos: Contiene todos los datos de los vuelos.
- Base: Las bases a dónde acuden las personas y los aviones.
- Función: Entidad dependiente de Personal, que describe la función de cada uno.
- Ciudades: Entidad dependiente de Vuelos, dónde se describe la ciudad destino y origen de cada vuelo.

4.1.3. Relación de cada entidad:

- Personal – Vuelo: Cada personal tiene varios vuelos y cada vuelo tiene mucho personal (N,M).
- Avión – Base: Cada avión tiene una base y cada base tiene varios aviones (1, N)

- Personal – Base: Cada personal tiene una base y cada base tiene mucho personal (1,N).
- Función – Persona: Cada función tiene mucho personal y cada personal tiene una función (N,1).
- Ciudades – Vuelos: Cada vuelo tiene un origen o un destino y cada ciudad tiene muchos vuelos (N,1).

4.2. Creación de Entidades:

Este paso es importante porque de este dependerá las conexiones y la creación de los objetos relacionales para poder trabajar en Symfony. Con el comando **php bin/console make:entity** podemos crear entidades.

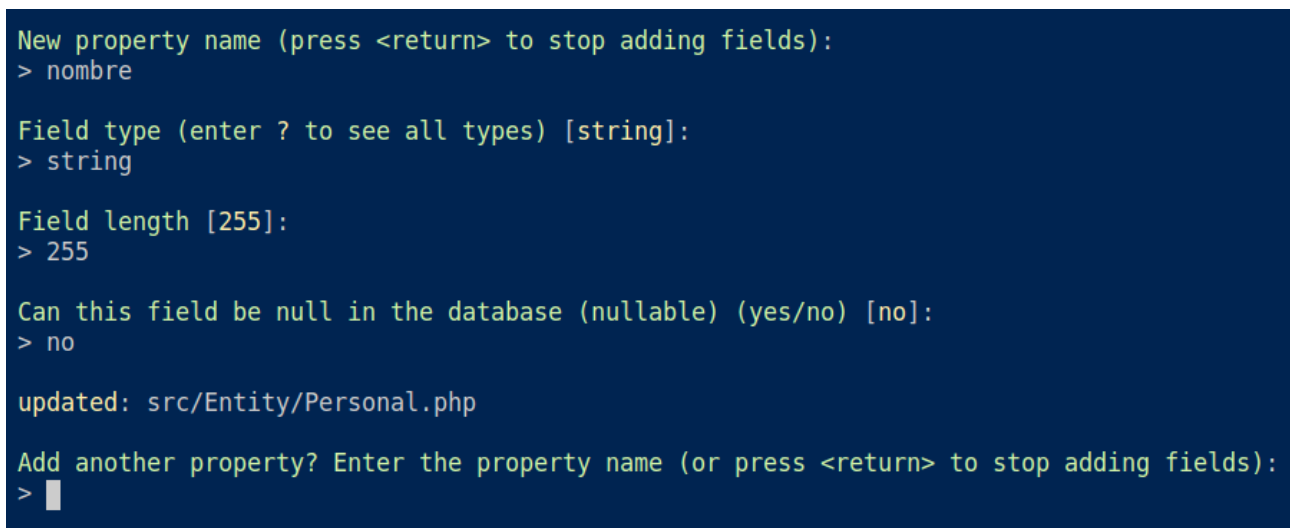


```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
cristian@pcrislinux: /var/www/aero_app$ php bin/console make:entity
PHP Warning: Module 'intl' already loaded in Unknown on line 0

```

Dentro de la guía para crear las entidades, nos pregunta por la propiedad que queremos crear y por el tipo de campo que es cada entidad.



```

New property name (press <return> to stop adding fields):
> nombre

Field type (enter ? to see all types) [string]:
> string

Field length [255]:
> 255

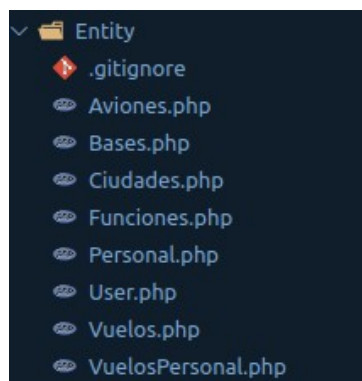
Can this field be null in the database (nullable) (yes/no) [no]:
> no

updated: src/Entity/Personal.php

Add another property? Enter the property name (or press <return> to stop adding fields):
> 

```

Este proceso se repite en todas las entidades. Luego de esto tendremos una carpeta con las entidades de esta manera:



4.2.1. Creación de relaciones:

El **make:entity** tiene la capacidad de crear relaciones entre cada entidad. Podemos crearlas según sea necesario para nuestras entidades cómo se ha detallado en pasos anteriores.

```
Field type (enter ? to see all types) [string]:
> relation

What class should this entity be related to?:
> Bases

What type of relationship is this?
-----
Type          Description
-----
ManyToOne     Each Personal relates to (has) one Bases.
               Each Bases can relate to (can have) many Personal objects

OneToMany     Each Personal can relate to (can have) many Bases objects.
               Each Bases relates to (has) one Personal

ManyToMany    Each Personal can relate to (can have) many Bases objects.
               Each Bases can also relate to (can also have) many Personal objects

OneToOne      Each Personal relates to (has) exactly one Bases.
               Each Bases also relates to (has) exactly one Personal.
-----

Relation type? [ManyToOne, OneToMany, ManyToMany, OneToOne]:
> ManyToOne

Is the Personal.bases_fk property allowed to be null (nullable)? (yes/no) [yes]:
> no
```

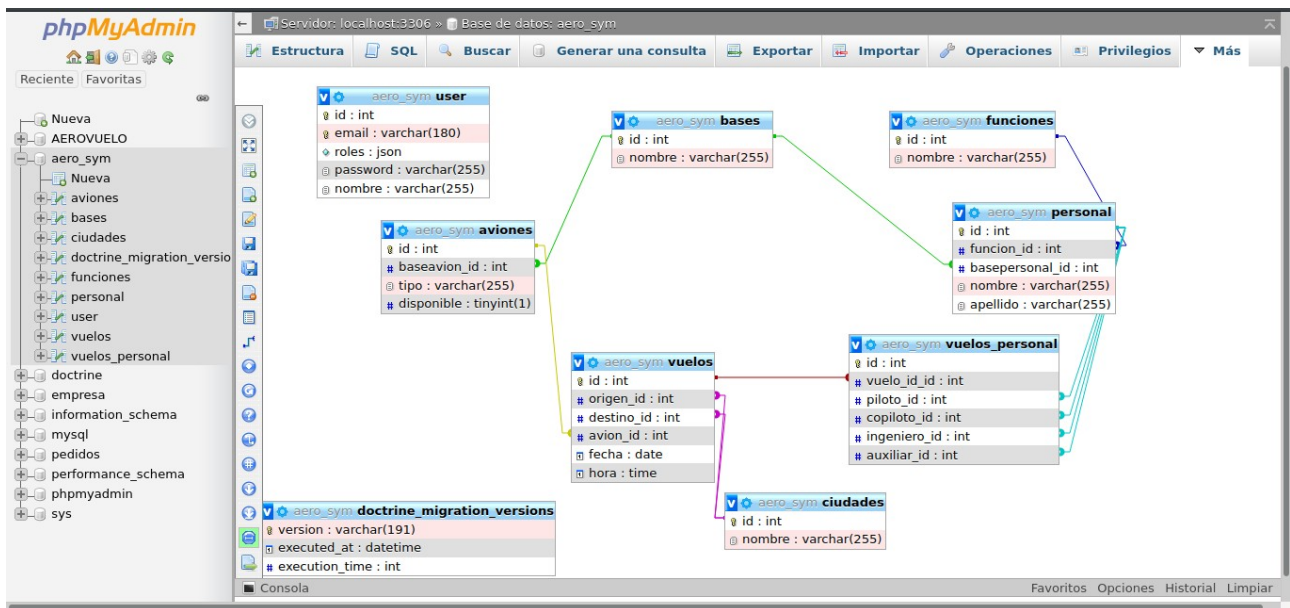
Figura 3: Relación Personal-Base

Con esto podremos tener relacionadas todas nuestras entidades. Luego de esto, tendremos que hacer las migraciones con el comando **php bin/console make:migration** y luego registrarlo en la base de datos con **php bin/console doctrine:migrate**.

```
cristian@pcrislinux:/var/www/aero_app$ php bin/console make:migration
```

4.3. Resultado en las bases de datos

Luego de esto, podremos ver los resultados en la base de datos con el modelo relacional y las entidades creadas.



Nótese que se ha creado una tabla adicional a las entidades que se llama `vuelos_personal`. Con esta tabla es posible hacer una relación de muchos a muchos.

4.4. Creación de datos base para la base de datos:

Antes de proceder a crear los datos de la base de datos, es importante hacer la creación de los usuarios que ingresarán a la aplicación y crear sus datos correspondientes, ya que estos se crean con el bundle de **make:fixtures** y este elimina y purga la base de datos antes de insertar los nuevos.

Así que para esto necesitaremos del comando **make:user** y crearemos los usuarios necesarios. Creará esto una entidad de usuarios con la clase Security, ya que aquí se crearán las contraseñas cifradas.

Luego de esto, creamos los datos para los usuarios con **make:fixtures**. Y este sería el contenido para la clase creada.

```

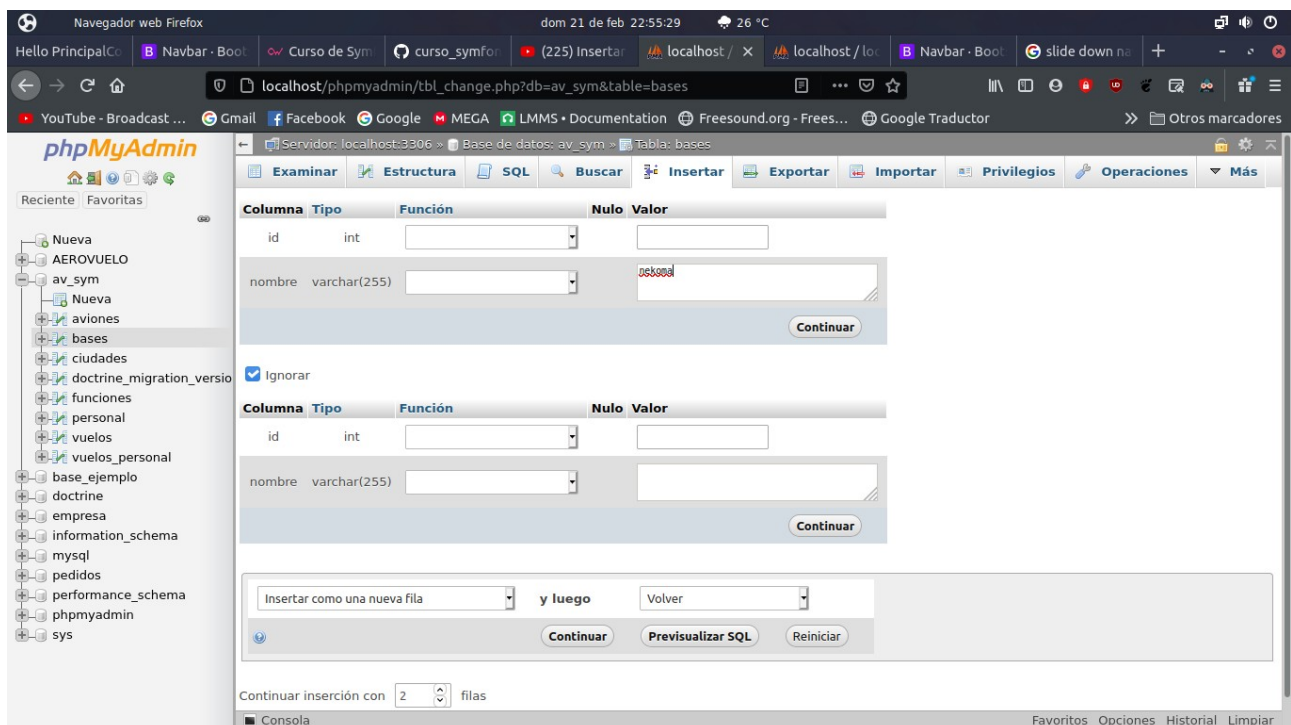
19     $this->userPasswordEncoder = $userPasswordEncoder;
20 }
21 public function load(ObjectManager $manager)
22 {
23     $usuario = new User();
24     $usuario->setEmail('cristian.admin@aero.es');
25     $usuario->setRoles(['ROLE_ADMIN']);
26     $usuario->setNombre('Cristian');
27     $usuario->setPassword(
28         $this->userPasswordEncoder->encodePassword($usuario, 'admin'));
29
30     $manager->persist($usuario);
31     $manager->flush();
32     $this->addReference(self::USUARIO_ADMIN_REFERENCIA, $usuario);
33
34     $usuario1 = new User();
35     $usuario1->setEmail('arley.user@aero.es');
36     $usuario1->setRoles(['ROLE_USER']);
37     $usuario1->setNombre('Arley');
38     $usuario1->setPassword(
39         $this->userPasswordEncoder->encodePassword($usuario1, '1234'));
40
41     $manager->persist($usuario1);
42     $manager->flush();
43     $this->addReference(self::USUARIO_USER_REFERENCIA, $usuario1);

```

Con esto, cargamos los datos a la base de datos con **load:fixtures** y ya tendríamos a los usuarios creados. Esta vez se han creado dos usuarios, uno con el rol de administrador y otro con el rol de usuario, que veremos a profundidad en la última parte del proyecto.

4.4.1. Inserción de datos:

Podemos crear datos desde código o directamente en el **phpMyAdmin**. Yo he elegido la segunda vía.



Al tener una interfaz gráfica amigable, se puede ingresar información sin mucha dificultad.

5. Creación de controladores, vistas y formularios con make:crud

El elemento más importante del Symfony es su capacidad de simplificar el trabajo de programación y el bundle de **make:crud** es capaz de crearnos las vistas necesarias a través del controlador que también crea por defecto.

Con el comando **php bin/console make:crud** solamente pasamos el nombre de la entidad a la cuál le crearemos el crud y automáticamente creará la carpeta de templates pertenecientes a la entidad, el controlador y dentro de este, todas las vistas que son necesarias como el ver cada uno de los datos de la tabla, editarlos y eliminarlos.

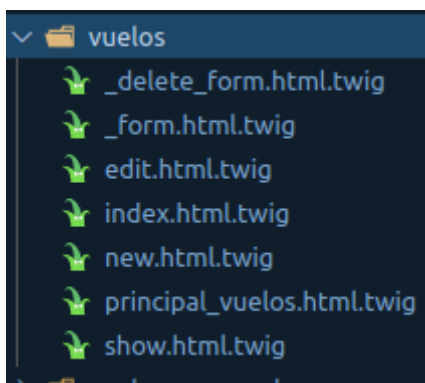


Figura 4: Vuelos Template

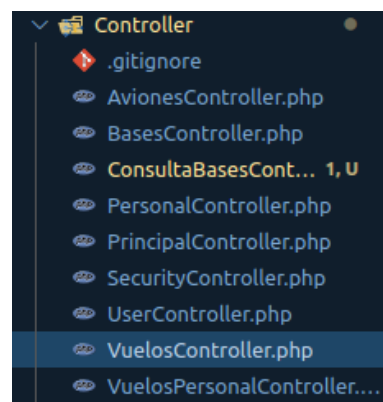
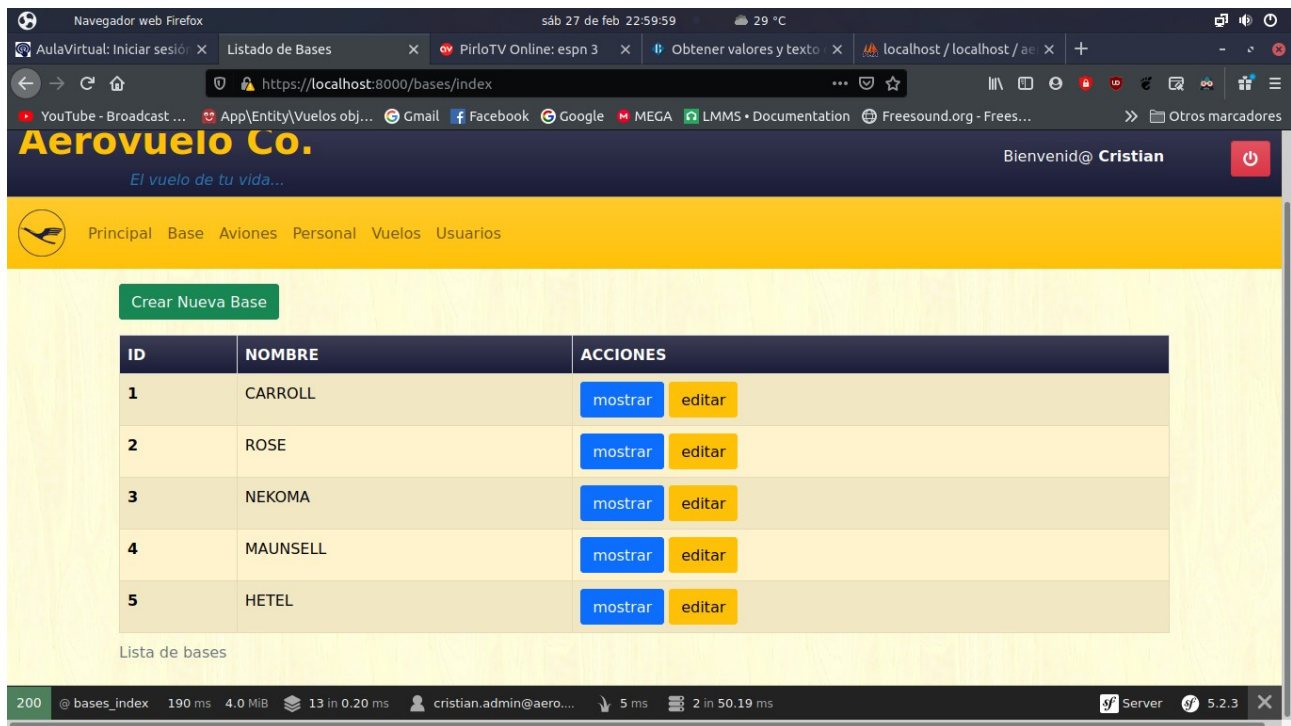


Figura 5: Carpeta de Controladores

Con esto ya podremos visualizar los resultados que nos da el controlador:



Cabe mencionar que en cada template he añadido propiedades y clases del Bootstrap para hacer vistosa la página que muestra cada template.

```
<table class="table table-bordered table-striped table-responsive mt-3">
  <caption>Lista de bases</caption>
  <thead class="text-uppercase bg-azul bg-gradient text-white">
    <tr>
      <th>Id</th>
      <th>Nombre</th>
      <th>acciones</th>
    </tr>
  </thead>
  <tbody class="text-uppercase table-warning">
    {% for basis in bases %}
      <tr>
        <td class="fw-bold">{{ basis.id }}</td>
        <td>{{ basis.nombre }}</td>
        <td>
```

Esto se repite en cada creación de **crud**, se añade elementos del Bootstrap y así tendríamos casi toda la aplicación realizada con las funcionalidades principales que son las de eliminar, editar y ver.

5.1. Paginación con KnpPaginator

Existe un bundle capaz de paginar nuestra lista de entidades por páginas. Debemos instalar primero el knppaginator desde su repositorio en composer con el siguiente comando.

```
composer require knplabs/knp-paginator-bundle
```

Luego de esto, en los servicios de la aplicación que se encuentra en la carpeta de los paquetes, se especifica la ruta y los temas que el paginador usará:

```
config > services.yaml > {} knp_paginator > {} template > filtration
49 App\Controller:
27     resource: '../src/Controller/'
28     tags: ['controller.service_arguments']
29
30 # add more service definitions when explicit configuration is needed
31 # please note that last definitions always *replace* previous ones
32
33 knp_paginator:
34     page range: 5 # number of links showed in the pagination menu (e.g: you have 10 pages, you are on page 5, it will show 4 previous and 4 next)
35     default options:
36         page name: page # page query parameter name
37         sort field name: sort # sort field query parameter name
38         sort direction name: direction # sort direction query parameter name
39         distinct: true # ensure distinct results, useful when ORM queries are using
40         filter field name: filterField # filter field query parameter name
41         filter value name: filterValue # filter value query parameter name
42     template:
43         pagination: '@KnpPaginator/Pagination/twitter_bootstrap_v4_pagination.html.twig' # sliding
44         sortable: '@KnpPaginator/Pagination/twitter_bootstrap_v4_font_awesome_sortable_link.html.twig' # font awesome
45         filtration: '@KnpPaginator/Pagination/twitter_bootstrap_v4_filtration.html.twig' # filters
```

Did you mean one of these?
make:unit-test Creates a new test class
make:user Creates a new security user class.

cristian@pcrislinux:/var/www/aerovuelo_app\$

Y por último, en cada controlador de nuestro proyecto, insertaremos el código necesario para que el paginador haga su función y en cada template lo mostraremos.

```
src > Controller > PersonalController.php > PHP Intelephense > PersonalController > index
21 * @Route("/", name="personal_principal")
22 */
23 public function principal()
24 {
25     return $this->render('personal/principal_personal.html.twig');
26 }
27 /**
28  * @Route("/index", name="personal_index", methods={"GET"})
29  */
30 public function index(EntityManagerInterface $em, PaginatorInterface $paginator, Request $request): Response
31 {
32     $sql = "SELECT p FROM App:Personal p";
33     $query = $em->createQuery($sql);
34     $pagination = $paginator->paginate(
35         $query, /* query NOT result */
36         $request->query->getInt('page', 1), /*page number*/
37         10 /*limit per page*/ You, 3 days ago * seventh commit
38     );
39     return $this->render('personal/index.html.twig', [
40         'pagination' => $pagination,
41     ]);
42 }
43 /**
44  * @Route("/new", name="personal_new", methods={"GET","POST"})
45  */
```

Did you mean one of these?
make:unit-test Creates a new test class
make:user Creates a new security user class.

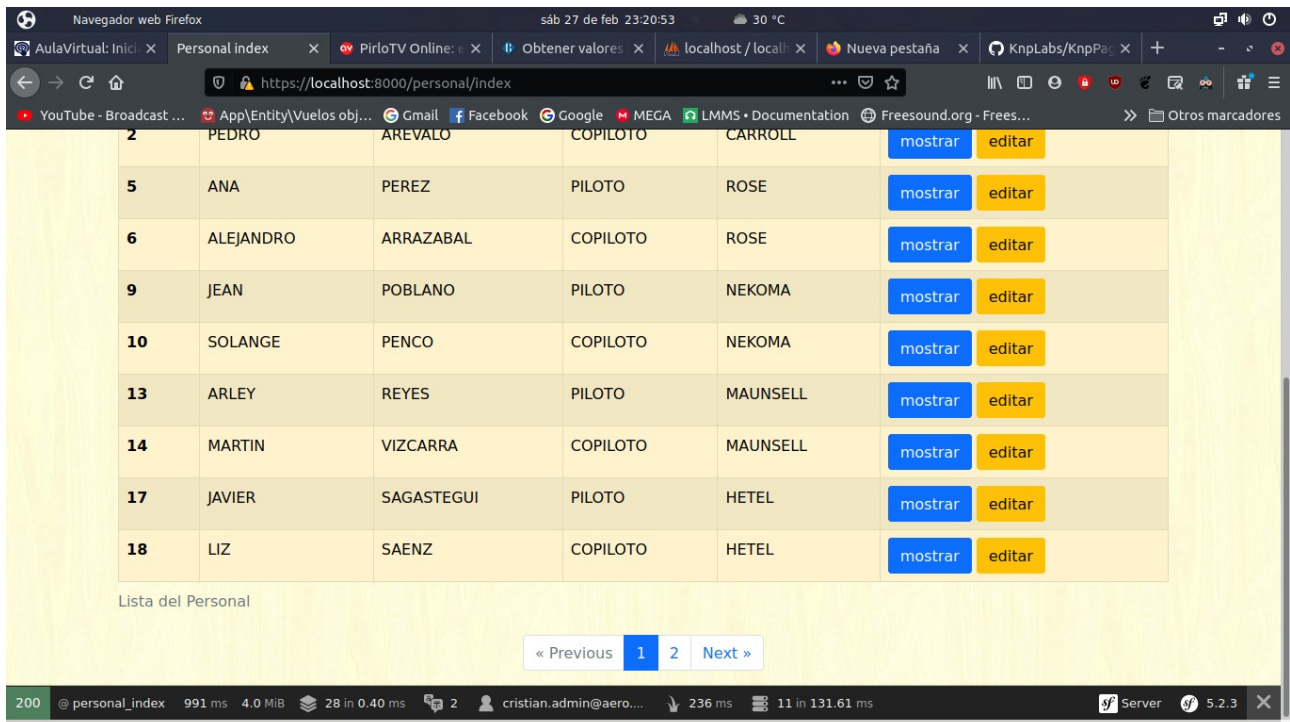
cristian@pcrislinux:/var/www/aerovuelo_app\$

Nótese que se llama a la clase del paginador y luego se obtiene una query sin los resultados, ya que el paginador hace la función de mostrar los resultados en la página según los parámetros de muestra (en este ejemplo, 10 filas por página).

Por último, en cada template se cambia el código original del **crud** por el del paginador para que se muestre por completo:

```
<div class="navigation">
  {{ knp_pagination_render(pagination) }}
</div>
```

Y estos son los resultados:



2	PEDRO	AREVALO	COPILOTO	CARROLL	mostrar	editar
5	ANA	PEREZ	PILOTO	ROSE	mostrar	editar
6	ALEJANDRO	ARRAZABAL	COPILOTO	ROSE	mostrar	editar
9	JEAN	POBLANO	PILOTO	NEKOMA	mostrar	editar
10	SOLANGE	PENCO	COPILOTO	NEKOMA	mostrar	editar
13	ARLEY	REYES	PILOTO	MAUNSELL	mostrar	editar
14	MARTIN	VIZCARRA	COPILOTO	MAUNSELL	mostrar	editar
17	JAVIER	SAGASTEGUI	PILOTO	HETEL	mostrar	editar
18	LIZ	SAENZ	COPILOTO	HETEL	mostrar	editar

Lista del Personal

« Previous 1 2 Next »

6. Consultas específicas

Cada entidad o vista de nuestro proyecto necesita de consultas específicas, para poder buscar bajo determinados parámetros. En este caso mostraré el ejemplo de la realización de una consulta con su respectiva vista, ya que este mismo proceso se repite para el resto de entidades.

Primero creamos un controlador para las consultas de cada entidad, en este caso, Bases:

```
ConsultaBasesCont... 1, U
```

Luego procedemos a crear las consultas necesarias en cada repositorio de cada entidad que esté relacionada con nuestra entidad principal, en este caso Bases está conectada con Personal y

Aviones:

```

src > Repository > PersonalRepository.php > PHP Intelephense > PersonalRepository > findByBase
You, seconds ago | 1 author (You)
15 class PersonalRepository extends ServiceEntityRepository
16 {
17     public function __construct(ManagerRegistry $registry)
18     {
19         parent::__construct($registry, Personal::class);
20     }
21
22     /**
23      * @return Personal[] Returns an array of Personal objects
24      */
25
26     public function findByBase($id)
27     {
28         return $this->createQueryBuilder('p')
29             ->andWhere('p.basepersonal = :id')
30             ->setParameter(['id', $id])
31             ->getQuery()
32             ->getResult();
33     }
34
35     /**
36      * public function findOneBySomeField($value): ?Personal
37      */
38

```

En el

repositorio de Personal, creamos una consulta específica que se le pasa por parámetro el **ID** de la base que se obtendrá en el formulario de búsqueda. Aquí entra en acción la potencia del doctrine para crear consultas devolviendo arrays con objetos de la clase que estamos buscando, muy similar a la lógica que usan los **json**.

El mismo procedimiento haremos para los aviones y luego en el controlador obtendremos los datos que se pasan por el formulario:

```

Visual Studio Code
sáb 27 de feb 23:23:38 30 °C
consulta_persona.html.twig - aerovuelo_app - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
AEROVUELO_APP
_form.html.twig
edit.html.twig
form_avibase.html.twig
index.html.twig
new.html.twig
principal_bases.html... M
show.html.twig
comunes
consulta_bases
consulta_avion.html.t... U
consulta_persona.ht... U
index.html.twig
personal
_delete_form.html.twig
_form.html.twig
edit.html.twig
index.html.twig
new.html.twig
principal_personal.html.twig
show.html.twig
principal
_carrusel.html.twig
templates > consulta_bases > consulta_persona.html.twig
1 {% extends "base.html.twig" %}
2 {% block title %} Consulta Bases {% endblock %}
3 {% block body %}
4
5 <form action={{path('lista_bases')}}>
6 <label for="personal">Personal: </label>
7 <select name="personal">
8     {% for base in bases %}
9         <option value={{base.id}}>{{base.nombre}} - {{base.id}}</option>
10     {% endfor %}
11 </select>
12 <input type="submit" value="Enviar">
13 </form>
14
15 {% if personal != null %}
16 <table class="table table-striped table-inverse table-responsive">
17     <caption>Lista de Personal en base {{base1}}</caption>
18     <thead>
19         <tr>
20             <th>ID</th>
21             <th>Nombre</th>
22             <th>Apellido</th>
23         </tr>
24     </thead>
25     <tbody>

```

Figura 6: Formulario

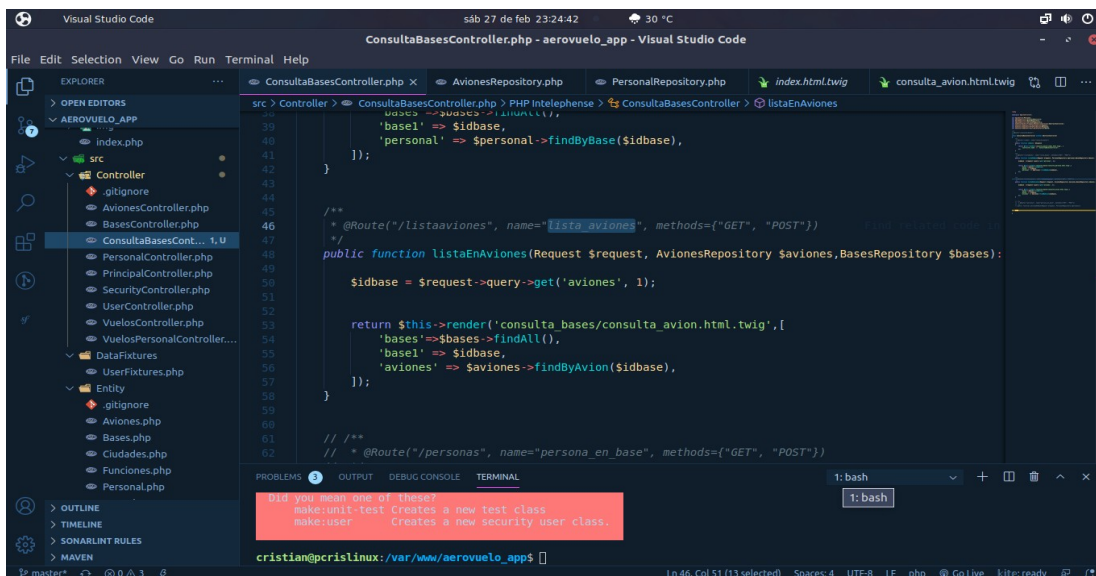
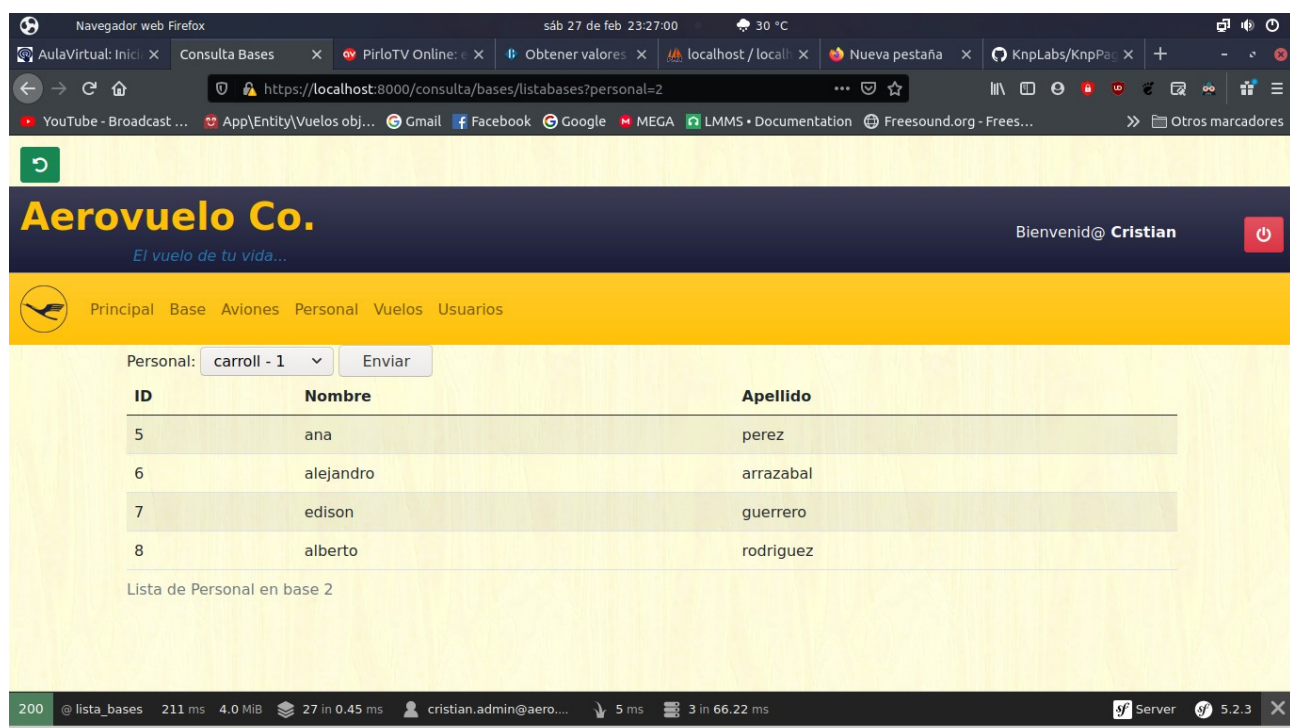


Figura 7: Controlador de Consulta

Y estos son los resultados:



7. Control de acceso

Symfony nos ofrece una capacidad de control de acceso y seguridad para diferenciar los roles de los usuarios y asegurarnos de que cada uno cumple con una función diferente.

Primero, debemos crear el login del usuario con el bundle **make:auth** y luego de esto nos creará automáticamente la vista del login, el controlador del login y un repositorio del mismo.

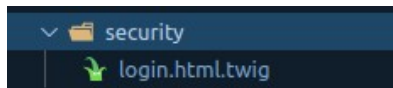


Figura 8: Template de Login

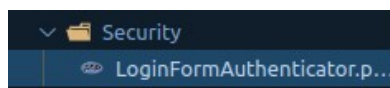


Figura 9: Repositorio

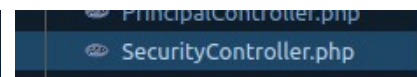
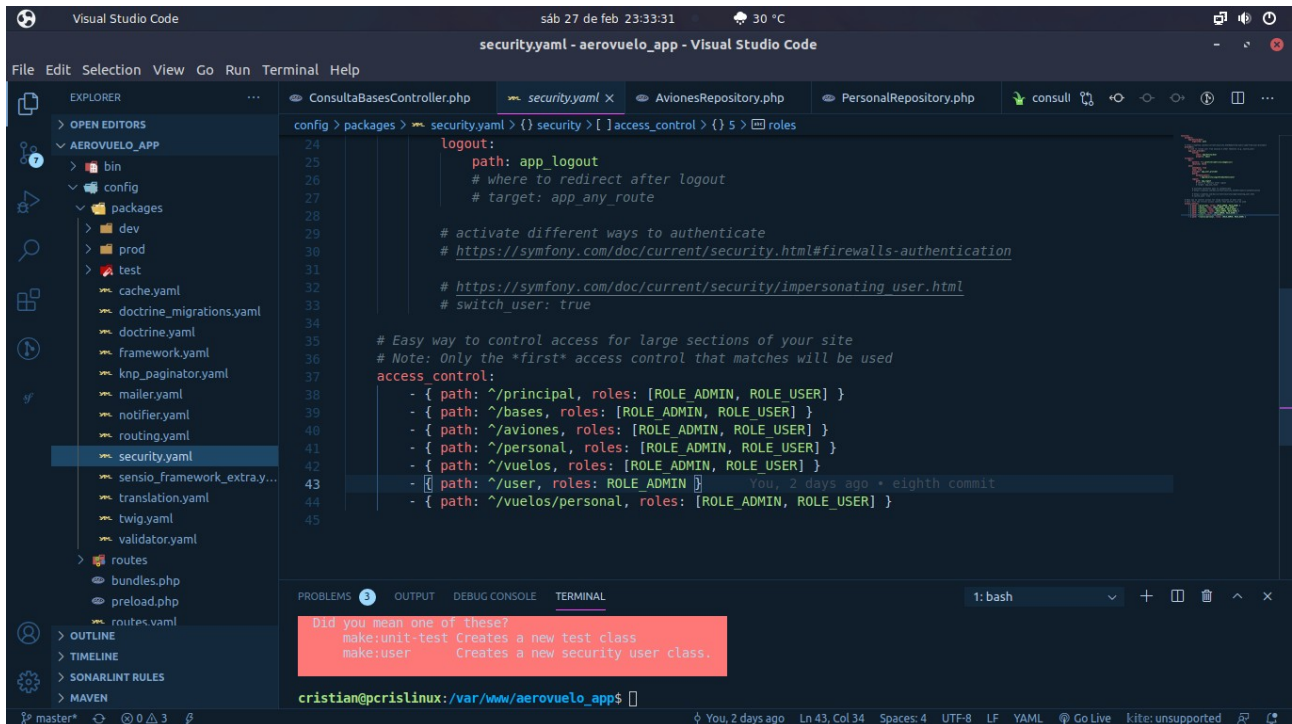


Figura 10: Controlador

Con esto, podemos ir al fichero **.yaml** de seguridad que se encuentra en las configuraciones de la aplicación y así poder indicar que usuario puede ver una vista determinada.



Y para ser más específicos, en cada template podemos agregar o quitar funciones según que usuario se ha logeado, así tendremos reservada la función de editar y de borrar para el admin y el usuario común solo podrá tener vistas específicas.



Y estos son los resultados:

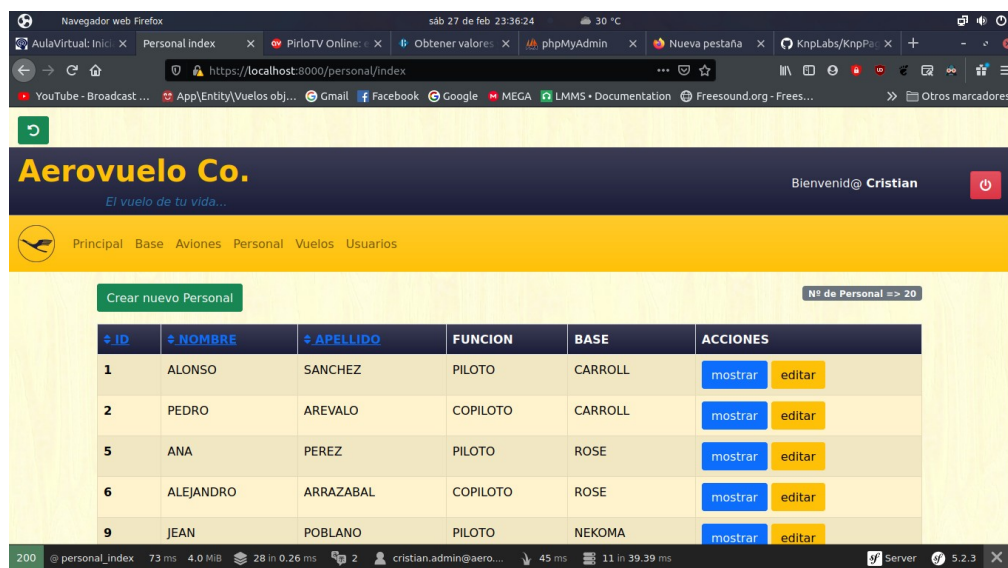


Figura 11: Vista de Admin

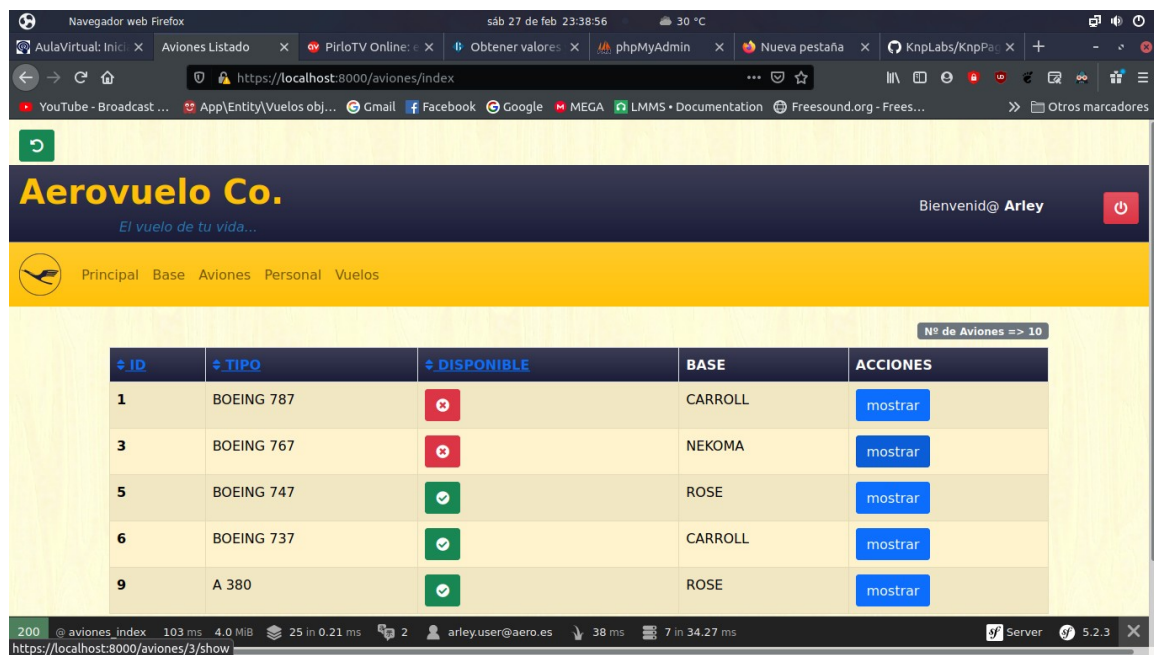


Figura 12: Vista de Usuario

8. Conclusiones

- ✓ Symfony desde mi experiencia es el framework más completo que existe para hacer proyectos web, no solo por cómo simplifica los caminos de programación, sino también por su capacidad y su potencia para administrar y visualizar las conexiones hacia las bases de datos.
- ✓ Algo que encuentro como un punto en contra es su curva de aprendizaje, que al inicio es demasiado tediosa por la infinidad de elementos y argumentos que se deben aprender para utilizar cada uno de sus bundles apropiadamente.
- ✓ En comparación a mi anterior proyecto, este ha sido mucho más rápido de hacer y mucho más completo, ya que Symfony me ha servido de funcionalidades que no veía posible implementar en mi anterior proyecto hecho con PHP.

- ✓ De esta manera podemos concluir con el proyecto, teniendo en frente uno que con un poco más de retoques perfectamente podría ser funcional para el uso en masas.

9. Bibliografia

<https://github.com/KnpLabs/KnpPaginatorBundle>

<https://symfony.com/>

<https://diego.com.es/symfony-y-doctrine>

<https://stackoverflow.com/>

10. Anexos:

10.1. Controladores:

10.1.1. Aviones:

```
<?php
```

```
namespace App\Controller;
```

```
use App\Entity\Aviones;
```

```
use App\Form\AvionesType;
```

```
use App\Repository\AvionesRepository;
```

```
use Knp\Component\Pager\PaginatorInterface;
```

```
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
```

```
use Symfony\Component\HttpFoundation\Request;
```

```
use Symfony\Component\HttpFoundation\Response;
```

```
use Symfony\Component\Routing\Annotation\Route;
```

```
use Doctrine\ORM\EntityManagerInterface;
```

```
/**
```

```
 * @Route("/aviones")
```

```
 */
```

```

class AvionesController extends AbstractController
{
/**
 * @Route("/", name="aviones_principal")
 */
public function principal(){
return $this->render('aviones/principal_bases.html.twig');
}
/**
 * @Route("/index", name="aviones_index", methods={"GET"})
 */
public function index(EntityManagerInterface $em, PaginatorInterface $paginator, Request
$request): Response
{
$dql = "SELECT a FROM App:Aviones a";
$query = $em->createQuery($dql);
$pagination = $paginator->paginate(
$query, /* query NOT result */
$request->query->getInt('page', 1), /*page number*/
5 /*limit per page*/
);
return $this->render('aviones/index.html.twig', [
'pagination' => $pagination,
]);
}

/**
 * @Route("/new", name="aviones_new", methods={"GET", "POST"})
 */
public function new(Request $request): Response
{
$avione = new Aviones();
$form = $this->createForm(AvionesType::class, $avione);
$form->handleRequest($request);

if ($form->isSubmitted() && $form->isValid()) {
$entityManager = $this->getDoctrine()->getManager();
$entityManager->persist($avione);
$entityManager->flush();

return $this->redirectToRoute('aviones_index');
}

return $this->render('aviones/new.html.twig', [
'avione' => $avione,
'form' => $form->createView(),
]);
}

```

```

/**
 * @Route("/{id}/show", name="aviones_show", methods={"GET"})
 */
public function show(Aviones $avione): Response
{
    return $this->render('aviones/show.html.twig', [
        'avione' => $avione,
    ]);
}

/**
 * @Route("/{id}/edit", name="aviones_edit", methods={"GET", "POST"})
 */
public function edit(Request $request, Aviones $avione): Response
{
    $form = $this->createForm(AvionesType::class, $avione);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $this->getDoctrine()->getManager()->flush();

        return $this->redirectToRoute('aviones_index');
    }

    return $this->render('aviones/edit.html.twig', [
        'avione' => $avione,
        'form' => $form->createView(),
    ]);
}

/**
 * @Route("/{id}", name="aviones_delete", methods={"DELETE"})
 */
public function delete(Request $request, Aviones $avione): Response
{
    if ($this->isCsrfTokenValid('delete', $avione->getId(), $request->request->get('_token'))) {
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->remove($avione);
        $entityManager->flush();
    }

    return $this->redirectToRoute('aviones_index');
}
}

```

10.1.2. Bases:

```
<?php
```

```
namespace App\Controller;

use App\Entity\Aviones;
use App\Entity\Bases;
use App\Entity\Personal;
use App\Form\AvionesType;
use App\Form\BasesType;
use App\Repository\AvionesRepository;
use App\Repository\BasesRepository;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;
//use Sensio\Bundle\FrameworkExtraBundle\Configuration\ParamConverter;

/**
 * @Route("/bases")
 */
class BasesController extends AbstractController
{
    /**
     * @Route("/", name="bases_principal")
     */
    public function principal() {
        return $this->render('bases/principal_bases.html.twig');
    }

    /**
     * @Route("/index", name="bases_index", methods={"GET"})
     */
    public function index(BasesRepository $basesRepository): Response
    {
        return $this->render('bases/index.html.twig', [
            'bases' => $basesRepository->findAll(),
        ]);
    }

    /**
     * @Route("/new", name="bases_new", methods={"GET", "POST"})
     */
    public function new(Request $request): Response
    {
        $basis = new Bases();
        $form = $this->createForm(BasesType::class, $basis);
    }
}
```

```

$form->handleRequest($request);

if ($form->isSubmitted() && $form->isValid()) {
    $entityManager = $this->getDoctrine()->getManager();
    $entityManager->persist($basis);
    $entityManager->flush();

    return $this->redirectToRoute('bases_index');
}

return $this->render('bases/new.html.twig', [
    'basis' => $basis,
    'form' => $form->createView(),
]);
}

/**
 * @Route("/{id}/show", name="bases_show", methods={"GET"})
 */
public function show(Bases $basis): Response
{
    return $this->render('bases/show.html.twig', [
        'basis' => $basis,
    ]);
}

/**
 * @Route("/{id}/edit", name="bases_edit", methods={"GET", "POST"})
 */
public function edit(Request $request, Bases $basis): Response
{
    $form = $this->createForm(BasesType::class, $basis);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $this->getDoctrine()->getManager()->flush();

        return $this->redirectToRoute('bases_index');
    }

    return $this->render('bases/edit.html.twig', [
        'basis' => $basis,
        'form' => $form->createView(),
    ]);
}

/**
 * @Route("/{id}", name="bases_delete", methods={"DELETE"})
 */

```

```

public function delete(Request $request, Bases $basis): Response
{
    if ($this->isCsrfTokenValid('delete',$basis->getId(), $request->request->get(' token'))) {
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->remove($basis);
        $entityManager->flush();
    }

    return $this->redirectToRoute('bases_index');
}
}

```

10.1.3. Personal:

```

<?php

namespace App\Controller;

use App\Entity\Personal;
use App\Form\PersonalType;
use App\Repository\PersonalRepository;
use Doctrine\ORM\EntityManagerInterface;
use Knp\Component\Pager\PaginatorInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

/**
 * @Route("/personal")
 */
class PersonalController extends AbstractController
{
    /**
     * @Route("/", name="personal_principal")
     */
    public function principal()
    {
        return $this->render('personal/principal_personal.html.twig');
    }
    /**
     * @Route("/index", name="personal_index", methods={"GET"})
     */
    public function index(EntityManagerInterface $em, PaginatorInterface $paginator, Request
    $request): Response
    {

```



```

$dql = "SELECT p FROM App:Personal p";
$query = $em->createQuery($dql);
$pagination = $paginator->paginate(
    $query, /* query NOT result */
    $request->query->getInt('page', 1), /*page number*/
    10 /*limit per page*/
);
return $this->render('personal/index.html.twig', [
    'pagination' => $pagination,
]);
}

/**
 * @Route("/new", name="personal_new", methods={"GET","POST"})
 */
public function new(Request $request): Response
{
    $personal = new Personal();
    $form = $this->createForm(PersonalType::class, $personal);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->persist($personal);
        $entityManager->flush();

        return $this->redirectToRoute('personal_index');
    }

    return $this->render('personal/new.html.twig', [
        'personal' => $personal,
        'form' => $form->createView(),
    ]);
}

/**
 * @Route("/{id}/show", name="personal_show", methods={"GET"})
 */
public function show(Personal $personal): Response
{
    return $this->render('personal/show.html.twig', [
        'personal' => $personal,
    ]);
}

/**
 * @Route("/{id}/edit", name="personal_edit", methods={"GET","POST"})
 */
public function edit(Request $request, Personal $personal): Response

```

```

{
$form = $this->createForm(PersonalType::class, $personal);
$form->handleRequest($request);

if ($form->isSubmitted() && $form->isValid()) {
$this->getDoctrine()->getManager()->flush();

return $this->redirectToRoute('personal_index');
}

return $this->render('personal/edit.html.twig', [
'personal' => $personal,
'form' => $form->createView(),
]);
}

/**
 * @Route("/{id}", name="personal_delete", methods={"DELETE"})
 */
public function delete(Request $request, Personal $personal): Response
{
if ($this->isCsrfTokenValid('delete'.$personal->getId(), $request->request->get('_token')))
{
$entityManager = $this->getDoctrine()->getManager();
$entityManager->remove($personal);
$entityManager->flush();
}

return $this->redirectToRoute('personal_index');
}
}

```

10.1.4. Vuelos

```

<?php

namespace App\Controller;

use App\Entity\Personal;
use App\Form\PersonalType;
use App\Repository\PersonalRepository;
use Doctrine\ORM\EntityManagerInterface;
use Knp\Component\Pager\PaginatorInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;

```

```

use Symfony\Component\Routing\Annotation\Route;

/**
 * @Route("/personal")
 */
class PersonalController extends AbstractController
{
    /**
     * @Route("/", name="personal_principal")
     */
    public function principal()
    {
        return $this->render('personal/principal_personal.html.twig');
    }

    /**
     * @Route("/index", name="personal_index", methods={"GET"})
     */
    public function index(EntityManagerInterface $em, PaginatorInterface $paginator, Request
$request): Response
    {
        $dql = "SELECT p FROM App:Personal p";
        $query = $em->createQuery($dql);
        $pagination = $paginator->paginate(
            $query, /* query NOT result */
            $request->query->getInt('page', 1), /*page number*/
            10 /*limit per page*/
        );
        return $this->render('personal/index.html.twig', [
            'pagination' => $pagination,
        ]);
    }

    /**
     * @Route("/new", name="personal_new", methods={"GET","POST"})
     */
    public function new(Request $request): Response
    {
        $personal = new Personal();
        $form = $this->createForm(PersonalType::class, $personal);
        $form->handleRequest($request);

        if ($form->isSubmitted() && $form->isValid()) {
            $entityManager = $this->getDoctrine()->getManager();
            $entityManager->persist($personal);
            $entityManager->flush();
        }

        return $this->redirectToRoute('personal_index');
    }
}

```

```

return $this->render('personal/new.html.twig', [
    'personal' => $personal,
    'form' => $form->createView(),
]);
}

/**
 * @Route("/{id}/show", name="personal_show", methods={"GET"})
 */
public function show(Personal $personal): Response
{
    return $this->render('personal/show.html.twig', [
        'personal' => $personal,
    ]);
}

/**
 * @Route("/{id}/edit", name="personal_edit", methods={"GET","POST"})
 */
public function edit(Request $request, Personal $personal): Response
{
    $form = $this->createForm(PersonalType::class, $personal);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $this->getDoctrine()->getManager()->flush();

        return $this->redirectToRoute('personal_index');
    }

    return $this->render('personal/edit.html.twig', [
        'personal' => $personal,
        'form' => $form->createView(),
    ]);
}

/**
 * @Route("/{id}", name="personal_delete", methods={"DELETE"})
 */
public function delete(Request $request, Personal $personal): Response
{
    if ($this->isCsrfTokenValid('delete', $personal->getId(), $request->request->get('_token')))
    {
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->remove($personal);
        $entityManager->flush();
    }

    return $this->redirectToRoute('personal_index');
}

```

```
}  
}
```

10.2. Repositorio

10.2.1. Aviones:

```
<?php
```

```
namespace App\Repository;
```

```
use App\Entity\Aviones;
```

```
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
```

```
use Doctrine\Persistence\ManagerRegistry;
```

```
/**
```

```
 * @method Aviones|null find($id, $lockMode = null, $lockVersion = null)
```

```
 * @method Aviones|null findOneBy(array $criteria, array $orderBy = null)
```

```
 * @method Aviones[] findAll()
```

```
 * @method Aviones[] findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)
```

```
*/
```

```
class AvionesRepository extends ServiceEntityRepository
```

```
{
```

```
    public function __construct(ManagerRegistry $registry)
```

```
    {
```

```
        parent::__construct($registry, Aviones::class);
```

```
    }
```

```
    public function consultaPorPersonalId($idForm){
```

```
        $dql = "SELECT persona.id, persona.apellido, persona.basepersonal FROM
```

```
App:Aviones as persona
```

```
WHERE :id = persona.id";
```

```
return $this->getEntityManager()
```

```
->createQuery($dql)->setParameter('id', $idForm)->getOneOrNullResult();
```

```
}
```

```
/**
```

```
 * @return Aviones[] Returns an array of Aviones objects
```

```
*/
```

```
    public function findByAvion($id)
```

```
    {
```

```
        return $this->createQueryBuilder('a')
```

```
->andWhere('a.baseavion = :id')
```

```
->setParameter('id', $id)
```

```
->getQuery()
```

```
->getResult()
```

```
;
```

```
}
```

```

/*
public function findOneBySomeField($value): ?Aviones
{
return $this->createQueryBuilder('a')
->andWhere('a.exampleField = :val')
->setParameter('val', $value)
->getQuery()
->getOneOrNullResult()
;
}
*/
}

```

10.2.2. Bases

```
<?php
```

```
namespace App\Repository;
```

```

use App\Entity\Aviones;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Persistence\ManagerRegistry;

```

```

/**
 * @method Aviones|null find($id, $lockMode = null, $lockVersion = null)
 * @method Aviones|null findOneBy(array $criteria, array $orderBy = null)
 * @method Aviones[] findAll()
 * @method Aviones[] findBy(array $criteria, array $orderBy = null, $limit = null, $offset =
null)
 */
class AvionesRepository extends ServiceEntityRepository
{
public function __construct(ManagerRegistry $registry)
{
parent::__construct($registry, Aviones::class);
}
public function consultaPorPersonalId($idForm){
$dql = "SELECT persona.id, persona.apellido, persona.basepersonal FROM
App:Aviones as persona
WHERE :id = persona.id";
return $this->getEntityManager()
->createQuery($dql)->setParameter('id', $idForm)->getOneOrNullResult();
}
}
/**
 * @return Aviones[] Returns an array of Aviones objects
 */

```

```

public function findByAvion($id)
{
return $this->createQueryBuilder('a')
->andWhere('a.baseavion = :id')
->setParameter('id', $id)
->getQuery()
->getResult()
;
}

/*
public function findOneBySomeField($value): ?Aviones
{
return $this->createQueryBuilder('a')
->andWhere('a.exampleField = :val')
->setParameter('val', $value)
->getQuery()
->getOneOrNullResult()
;
}
*/
}

```

10.2.3. Personal

```
<?php
```

```

namespace App\Repository;

use App\Entity\Aviones;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Persistence\ManagerRegistry;

/**
 * @method Aviones|null find($id, $lockMode = null, $lockVersion = null)
 * @method Aviones|null findOneBy(array $criteria, array $orderBy = null)
 * @method Aviones[] findAll()
 * @method Aviones[] findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)
 */
class AvionesRepository extends ServiceEntityRepository
{
public function __construct(ManagerRegistry $registry)
{
parent::__construct($registry, Aviones::class);
}
public function consultaPorPersonalId($idForm){

```

```

$dql = "SELECT persona.id, persona.apellido, persona.basepersonal FROM
App:Aviones as persona
WHERE :id = persona.id";
return $this->getEntityManager()
->createQuery($dql)->setParameter('id', $idForm)->getOneOrNullResult();
}
/**
 * @return Aviones[] Returns an array of Aviones objects
 */
public function findByAvion($id)
{
return $this->createQueryBuilder('a')
->andWhere('a.baseavion = :id')
->setParameter('id', $id)
->getQuery()
->getResult()
;
}

/**
public function findOneBySomeField($value): ?Aviones
{
return $this->createQueryBuilder('a')
->andWhere('a.exampleField = :val')
->setParameter('val', $value)
->getQuery()
->getOneOrNullResult()
;
}
*/
}

```

10.2.4. Vuelos

```
<?php
```

```
namespace App\Repository;
```

```

use App\Entity\Aviones;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Persistence\ManagerRegistry;

```

```

/**
 * @method Aviones|null find($id, $lockMode = null, $lockVersion = null)
 * @method Aviones|null findOneBy(array $criteria, array $orderBy = null)
 * @method Aviones[] findAll()
 * @method Aviones[] findBy(array $criteria, array $orderBy = null, $limit = null, $offset =
null)
 */

```



```

class AvionesRepository extends ServiceEntityRepository
{
public function __construct(ManagerRegistry $registry)
{
parent::__construct($registry, Aviones::class);
}
public function consultaPorPersonalId($idForm){
$dql = "SELECT persona.id, persona.apellido, persona.basepersonal FROM
App:Aviones as persona
WHERE :id = persona.id";
return $this->getEntityManager()
->createQuery($dql)->setParameter('id', $idForm)->getOneOrNullResult();
}
/**
 * @return Aviones[] Returns an array of Aviones objects
 */
public function findByAvion($id)
{
return $this->createQueryBuilder('a')
->andWhere('a.baseavion = :id')
->setParameter('id', $id)
->getQuery()
->getResult()
;
}

/**
public function findOneBySomeField($value): ?Aviones
{
return $this->createQueryBuilder('a')
->andWhere('a.exampleField = :val')
->setParameter('val', $value)
->getQuery()
->getOneOrNullResult()
;
}
*/
}

```