Controladores

Conocimientos que se presuponen:

- Crear controladores a mano
- Renderizar una plantilla de Twig
- Pasar parámetros a la plantilla de Twig
- Generar urls
- Redireccionar a otras rutas o a una url

Nuevo en Symfony 4

En symfony 4 existe un comando para generar un controlador

> *php bin/console make:controller NombreController*

También existe un comando para generar un CRUD completo a partir de una entidad de Doctrine:

> *php bin/console make:crud Producto*

Nota: Estos comandos forman parte de MakerBundle

Controller y AbstractController

Una clase controladora en symfony puede extender de Controller o de AbstractController.

La única diferencia es que con AbstractController NO puedes acceder a los servicios directamente a través de los métodos $this->get() o $this->container->get(). Esto obliga a escribir código algo más robusto para acceder a los servicios.

Excepciones

En cualquier controlador en el que lancemos una excepción, Symfony generará una respuesta con código 500.

```
public function index()
{
    // ...
    throw new \Exception('Algo ha ido mal');
}
```

Si estamos en modo debug, además se mostrará una traza completa del error.

Páginas 404

También podemos hacer Symfony responde con un 404 Not Found desde un controlador.

A pesar de que la ruta exista, quizás el recurso que se pide no existe, y en ese caso se debe devolver al navegador una respuesta 404.

Hay dos formas de hacerlo. La primera es lanzando una excepción especial de Symfony: **NotFoundHttpException**.

```
use Symfony\Component\HttpKernel\Exception\NotFoundHttpExc
```

```
eption;

// ...
public function index()
{
    $product = ...;
    if (!$product) {
        throw new NotFoundHttpException('El producto solic
itado no existe');
    }

    return $this->render(...);
}
```

La segunda es un atajo de la primera, y consiste en utilizar la función **createNotFoundException()**.

```
use Symfony\Component\HttpKernel\Exception\NotFoundHttpExc
eption;

// ...
public function index()
{
    $product = ...;
    if (!$product) {
        throw $this->createNotFoundException('El producto
solicitado no existe');
```

```
    }

    return $this->render(...);
}
```

Ambas formas son equivalentes.

Personalizar las páginas de error

Las páginas de error del entorno de producción se pueden personalizar.

Cuando se carga una página de error, se utiliza un controlador interno (ExceptionController) para renderizar una plantilla de Twig.

Este controlador elige la plantilla según el código de error HTTP y el formato de la request siguiendo el siguiente algoritmo:

1. Buscar una plantilla con el nombre errorXXX.format.twig (por ejemplo: error404.json.twig o error500.html.twig).

2. Si no existe la plantilla buscada en el punto anterior, se busca una plantilla de error genérica, sin código: error.json.twig, error.xml.twig…

3. Si tampoco existe una plantilla con ese nombre, renderiza la plantilla genérica error.html.twig.

```
templates/
└─ bundles/
   └─ TwigBundle/
      └─ Exception/
         ├─ error404.html.twig
         ├─ error403.html.twig
         ├─ error.html.twig
         ├─ error404.json.twig
         ├─ error403.json.twig
         └─ error.json.twig
```

Las plantillas se buscan en el directorio Exception de las plantillas del bundle TwigBundle, así que, según las normas de sobrescribir plantillas, deberíamos ubicarlas en:
templates/bundles/TwigBundle/Exception/

Visualizar páginas de error

Mientras estamos en desarrollo, no podemos ver cómo quedan las páginas de error que estamos diseñando, porque saldrían las páginas de error del entorno de desarrollo con toda la traza del error.

Twig dispone de unas rutas especiales que podemos utilizar para este propósito:

```
# config/routes/dev/twig.yaml
_errors:
```

```
    resource: '@TwigBundle/Resources/config/routing/errors
.xml'
    prefix:   /_error
```

Con estas rutas, podemos utilizar las siguientes urls para visualizar las páginas de error:

```
http://localhost/index.php/_error/{statusCode}
http://localhost/index.php/_error/{statusCode}.{format}
```

Sesión

Symfony cuenta con un servicio para almacenar la información relacionada con la sesión del usuario

Para utilizarla, debemos activarla en la configuración del framework

```
# config/packages/framework.yaml
framework:
    # ...

    session:
        # The native PHP session handler will be used
        handler_id: ~
    # ...
```

Para acceder a la sesión basta con tipar un argumento con el tipo **SessionInterface**

```
use Symfony\Component\HttpFoundation\Session\SessionInterf
ace;

public function index(SessionInterface $session)
{
    // store an attribute for reuse during a later user re
quest
```

```
    $session->set('foo', 'bar');

    // get the attribute set by another controller in anot
her request
    $foobar = $session->get('foobar');

    $foobar = $session->has('foobar');

    // use a default value if the attribute doesn't exist
    $filters = $session->get('filters', array());

    //Destruye la sesión y crea otra nueva
    $foobar = $session->invalidate();

    //Borra todos los atributos de la sesión actual
    $foobar = $session->clear();
}
```

Acceder a la sesión desde Twig

Recordemos que desde twig tenemos acceso a la variable app, que no permite acceder a la request, la sesión, etc

- app.user
- app.request
- app.session
- app.environment

- app.debug

Los mensajes flash

Los mensajes flash son internamente atributos de sesión que pueden ser utilizados una única vez. Desaparecen de la sesión automáticamente cuando recuperamos su valor.

Son útiles para mostrar notificaciones al usuario.

Veamos un ejemplo:

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

public function updateAction(Request $request)
{
    // ...

    if ($form->isSubmitted() && $form->isValid()) {
        // do some sort of processing

        $this->addFlash(
            'notice',
            'Your changes were saved!'
        );

        return $this->redirectToRoute(...);
    }
```

```
    return $this->render(...);
}
```

En una plantilla concreta o incluso en una plantilla base, podemos acceder a los mensajes flash utilizando app.flashes().

```
{# app/Resources/views/base.html.twig #}

{# Podemos recuperar solamente un tipo de mensajes #}
{% for message in app.flashes('notice') %}
    <div class="flash-notice">
        {{ message }}
    </div>
{% endfor %}

{# ...o recuperarlos todos #}
{% for label, messages in app.flashes %}
    {% for message in messages %}
        <div class="flash-{{ label }}">
            {{ message }}
        </div>
    {% endfor %}
{% endfor %}
```

NOTA: Es muy común utilizar tres claves de mensajes flash: notice, warning y error, pero realmente podemos utilizar los que queramos sin ningún problema.

NOTA: Existe el método peek() para obtener los mensajes SIN ELIMINARLOS de la sesión.

```
{% for message in app.peek('notice') %}
```

El objeto Request

Toda la información sobre la petición, es accesible desde el objeto Request.

Basta con incluirlo como argumento del controlador para tener acceso a él.

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

public function index(Request $request)
{
    $request->isXmlHttpRequest(); // is it an Ajax request
?

    $request->getPreferredLanguage(array('en', 'fr'));

    // obtener variables enviadas por GET y por POST respe
ctivamente
    $request->query->get('page');
    $request->request->get('page');

    // obtener variables $_SERVER
    $request->server->get('HTTP_HOST');

    // obtener una instancia de la clase UploadedFile con
el fichero enviado
```

```
    $request->files->get('fichero');

    // obtener el valor de una COOKIE
    $request->cookies->get('PHPSESSID');

    // obtener una cabecera HTTP
    $request->headers->get('host');
    $request->headers->get('content_type');
}
```

El objeto Response

Un controlador en Symfony tiene un objetivo principal: Devolver un objeto response.

Ya sea una página html, o una respuesta en formato json, o una página de error, o un archivo para descargar... deberíamos acabar siempre haciendo un return de un objeto de tipo Response.

Un ejemplo sencillo sería el siguiente:

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

...

$response = new Response(
    'Contenido de la respuesta',
    Response::HTTP_OK,
    array('content-type' => 'text/html')
);

return $response;
```

El objeto response tiene métodos para establecer los valores del content, del código de estado, de los headers...

```
$response->setContent('<h1>Hello World</h1>');

$response->headers->set('Content-Type', 'text/plain');

$response->headers->setCookie(new Cookie('foo', 'bar'));

$response->setStatusCode(Response::HTTP_NOT_FOUND);

$response->setCharset('ISO-8859-1');
```

NOTA: Desde la versión 3.1, no se puede configurar el charset por defecto de las respuestas en el config.yml. Se debe hacer en la clase AppKernel. (Symony utiliza por defecto UTF-8).

```
class AppKernel extends Kernel
{
    public function getCharset()
    {
        return 'ISO-8859-1';
    }
}
```

Renderizando plantillas

Cuando utilizamos el método $this->render(), internamente symfony construye y nos devuelve un objeto Response.

Al hacer return $this->render() estamos por lo tanto cumpliendo la norma de symfony de hacer un return de un objeto Response.

Ejemplo de una respuesta JSON

El siguiente ejemplo muestra cómo se haría una respuesta json:

```
$response = new Response();
$response->setContent(json_encode(array(
    'data' => 123,
)));
$response->headers->set('Content-Type', 'application/json'
);
return $response
```

Objetos que extienden de Response

Symfony dispone de objetos que extienden de Response para facilitar tipos de respuesta muy comunes:

- RedirectResponse
- JsonResponse
- BinaryFileResponse
- StreamedResponse

Además, si extendemos nuestra clase de Controller o de AbstractController, tenemos disponibles funciones helper que fa[illegible]a

generación de los objetos Response correspondientes:

- $this->redirect('http://symfony.com/doc');
- $this->json(array('data' => 123));
- $this->file('/path/to/some_file.pdf');

https://symfony.com/doc/current/components/http_foundation.html#compo
http-foundation-serving-files

https://symfony.com/doc/current/components/http_foundation.html#strean
response

Depuración en los controladores

En el componente **VarDumper** existe una función llamada *dump()* muy útil para la depuración de variables tanto en twig como en los controladores.

Para utilizar esta función antes de nada tenemos que asegurarnos de tener el componente instalado:

> *composer require var-dumper*

Ahora ya podemos utilizarlo.

// src/Controller/ArticleController.php
namespace App\Controller;

// ...

class ArticleController extends Controller
{
public function recentList()
{
$articles = ...;
dump($articles);

```
    // ...
}
```

```
}
```

La función dump() renderiza el valor de la variable en la barra de depuración.