ESTRUCTURA DE SYMFONY

MODELO, VISTA, CONTROLADOR

MVC

El framework de symfony utiliza la estructura MVC.

Esta arquitectura permite dividir nuestras aplicaciones en tres grandes capas

a) Modelado: Es el responsable de la conexión a la base de datos y la manipulación de los datos mismos. Esta capa esta pensada para trabajar con los datos y para obtenerlos. Es decir tiene la parte lógica de la aplicación.

Toda esta parte en la aplicación se desarrolla en el directorio src.

MVC

b) La vista

Es el canal de salida, el puente para enviar los datos que devuelve el modelo al usuario que accede al sistema. Por tanto es todo lo que se refiera a la visualización de la información, el diseño, colores, estilos y la estructura visual en sí de nuestras páginas

Estas vistas están reunidas en las carpetas templates. Las vistas son archivos Twig, que es un lenguaje muy sencillo de plantillas.

MVC

C) El controlador

Atiende las peticiones o eventos, llama a la lógica de los modelos y retorna resultados a través de la vista.
Es decir, trabaja de intermediario entre las dos capas.

Los controladores agrupan todos los programas PHP que coordinará la aplicación.


funcionamiento de MVC
entrega 6
solicita
1
2
invoca
CONTROLADOR
procesa y envía
5
4
envía
MODELO
VISTA
pide
retorna
Lógica
Layout
3
procesa
En el modelo que se tratará en este libro, la lógica de la vista recibe la información provista por el controlador. Trae el layout (diseño) y lo procesa con la información recibida, antes de entregarlo al usuario. Otros modelos, pueden realizar esto, íntegramente a través del controlador, pero aquí no lo trataremos por considerar que resta mantenibilidad al sistema.

MVC

El funcionamiento básico del patrón MVC, puede resumirse en:

- ✓El usuario realiza una petición
- ✓El controlador captura el evento Hace la llamada al modelo/modelos correspondientes efectuando las modificaciones pertinentes sobre el modelo
- ✓El modelo será el encargado de interactuar con la base de datos, y retornará esta información al controlador.
- ✓Se refiere a la visualización de la información, el diseño, colores, estilos y la estructura visual en sí de nuestras páginas.

INTRODUCCIÓN A SYMFONY

Symfony optimiza el desarrollo de aplicaciones Web, proporcionando herramientas para agilizar aplicaciones complejas.

Reutiliza conceptos y desarrollos exitosos de terceros y los integra como librerías para ser utilizados por nosotros.

Entre ellos encontramos que integra plenamente uno de los **frameworks ORM(Object Relational Mapping o Mapeo Objeto-Relacional)** más importantes dentro de los existentes para PHP llamado **Doctrine**, el cual es el encargado de la comunicación con la base de datos, permitiendo un control casi total de los datos sin importar si estamos hablando de MySQL, SQLserver, u Oracle, entre otros motores ya que la mayoría de las sentencias SQL no son generadas por el programador sino por el mismo Doctrine.

INTRODUCCIÓN A SYMFONY

Otro ejemplo de esto es la inclusión del **framework Twig**, un poderoso motor de plantillas que nos permite separar el código PHP del HTML permitiendo una amplia gama de posibilidades y por sobre todo un extraordinario orden para nuestro proyecto

También contamos con las instrucciones denominadas de consola en la terminal diciéndole a Symfony que nos genere lo necesario para lo que le estamos pidiendo. (entity,....etc)

INTRODUCCIÓN A SYMFONY

Otra de las funcionalidades más interesantes, es que contiene un subframework para trabajar con formularios. Con esto, creamos una clase orientada a objetos que representa al formulario HTML y una vez hecho lo mostramos y ejecutamos.

Es decir que no diseñamos el formulario con HTML sino que lo programamos utilizando herramientas del framework.

Esto nos permite tener en un lugar ordenados todos los formularios de nuestra aplicación incluyendo las validaciones realizadas en el lado del servidor, ya que symfony implementa objetos validadores muy sencillos y potentes para asegurar la seguridad de los datos introducidos por los usuarios

PRIMERAS PÁGINAS

Para crear una página tenemos que tener como mínimo:

- ✓ Creación de una acción (action): La lógica necesaria para la página.

  Y corresponde al Controlador en arquitectura MVC.

- ✓ Asignación de una ruta: Es una URL asignada a la página, para que el controlador frontal pueda acceder a ella.
- ✓ Creación de la plantilla (template): La estructura de nuestra página. Corresponde a la Vista en arquitectura MVC.

CREAR LOS CONTROLADORES

Crear un controlador: dentro del MVC tiene el rol de comunicar la vista con el modelo, de manera que recibirá las peticiones del usuario, ejecutará el código necesario parar resolver la petición, y enviar a el modelo y la vista con la información para representar y para devolverla al usuario.

Se encuentran en la carpeta src/Controller. Por ahora está vacía.

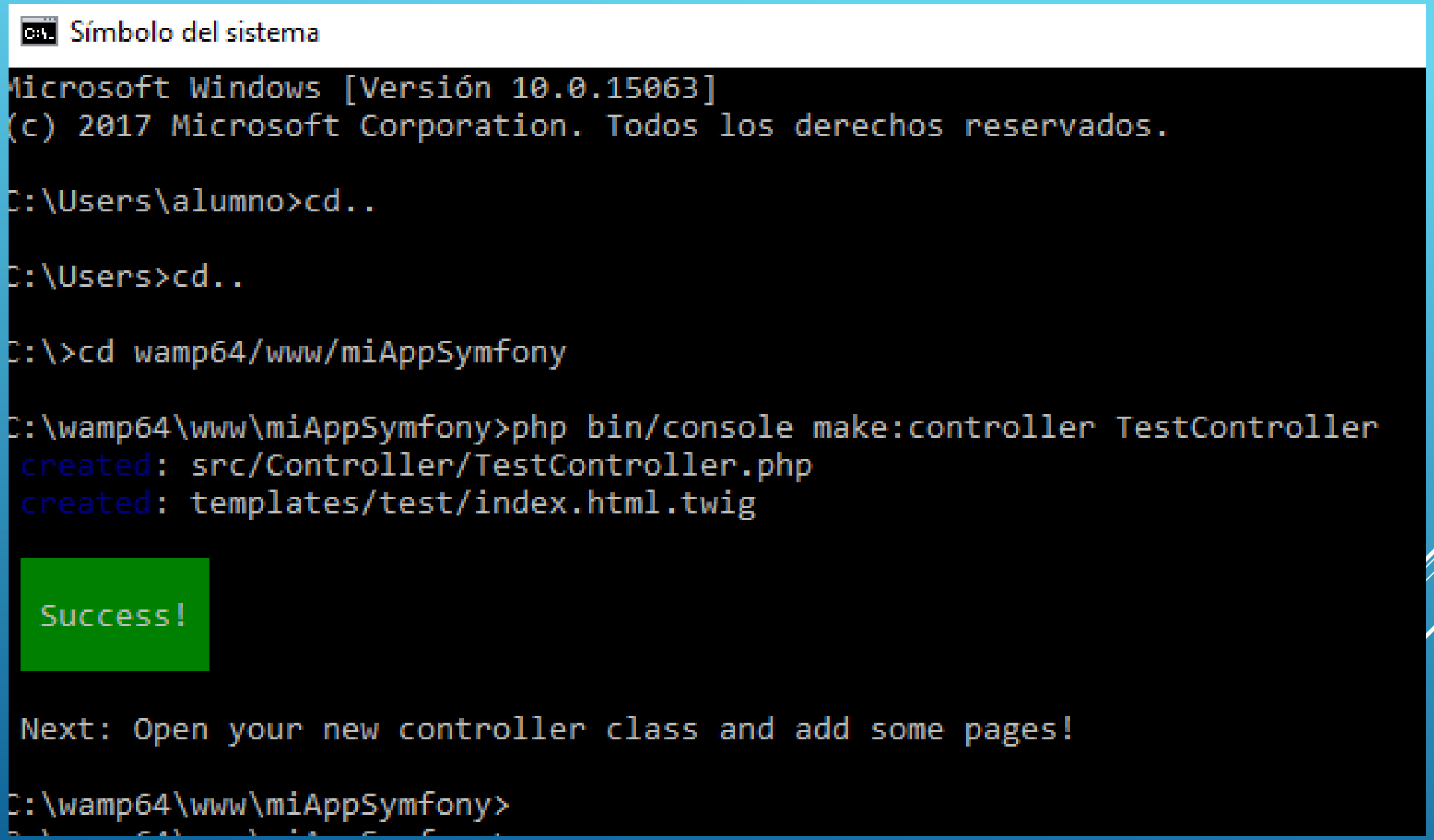
Los controladores son clases contenidas en un archivo que debe tener el mismo nombre.

CREAR LOS CONTROLADORES

Todos los nombres de los controladores deben terminar en Controller. Hay que prestas atención a las mayúsculas ya que son importantes para evitar tediosos errores (cada palabra empieza por mayúsculas y no hay separadores)

Es posible crear un controlador manualmente pero lo mejor es usar la consola de symfony en línea de comandos.

```
php bin/console make:controller NombreControlador
```


Símbolo del sistema
Microsoft Windows [Versión 10.0.15063]
(c) 2017 Microsoft Corporation. Todos los derechos reservados.
C:\Users\alumno>cd..
C:\Users>cd..
C:\>cd wamp64/www/miAppSymfony
C:\wamp64\www\miAppSymfony>php bin/console make:controller TestController
created: src/Controller/TestController.php
created: templates/test/index.html.twig
Success!
Next: Open your new controller class and add some pages!
C:\wamp64\www\miAppSymfony>

LOS CONTROLADORES

Como vemos en la pantalla anterior nos indica que el controlador se ha creado en el directorio src/Controller.

Y en el directorio template/Test ha hecho una vista sencilla para darle un formato de salida.

Vamos a ver el código que se ha generado en la plantilla y en este controlador

```php
<?php
namespace App\Controller;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;


class TestController extends AbstractController
{    #[Route('/test', name: 'app_test')]

 public function index(): Response
  {
       return $this->render('test/index.html.twig', [            'controller_name' => 'TestController',        ]);
  }
}
```

LOS CONTROLADORES

La primera línea es el namespace de la clase.

Las tres líneas siguientes son las clases que vamos a utilizar en este controlador

AbstractController es la clase de la que heredan todos los controladores de manera predeterminada. Nos permite utilizar una serie de métodos comunes a todos los controladores.

Response es la clase qué será el tipo de respuesta enviada por el controlador(como podemos ver la función index devuelve un tipo response:

public function index(): Response{…..}

LOS CONTROLADORES

Route es la clase que gestiona las rutas del controlador, las rutas son el vinculo entre l solicitud enviada por el usuario y el nombre del método que se va a ejecutar en el controlador.

Route utiliza lo que se denomina anotaciones(Parecen comentarios pero no lo son):

#[Route('/test', name: 'app_test')]

LOS CONTROLADORES

Las anotaciones siempre están justo encima de la declaración de la función a la que gobierna.

El parámetro **name** asigna el nombre a esa ruta, este **nombre debe ser único**, ya que ya que es un nombre lógico, y es como lo reconoce el servidor.

El método index() ejecuta el método render (a través del AbstractController).

Con esto permite ejecutar la vista llamada index.html.twig de la carpeta templates/test.

Transmite a la vista una variable controller_name que contiene el valor de TestController.

Si quisiéramos retornar valores desde nuestro propio controlador sería con:

return new Response(.....) en este caso el controlador puede crear y de volver un objeto de Response como respuesta.

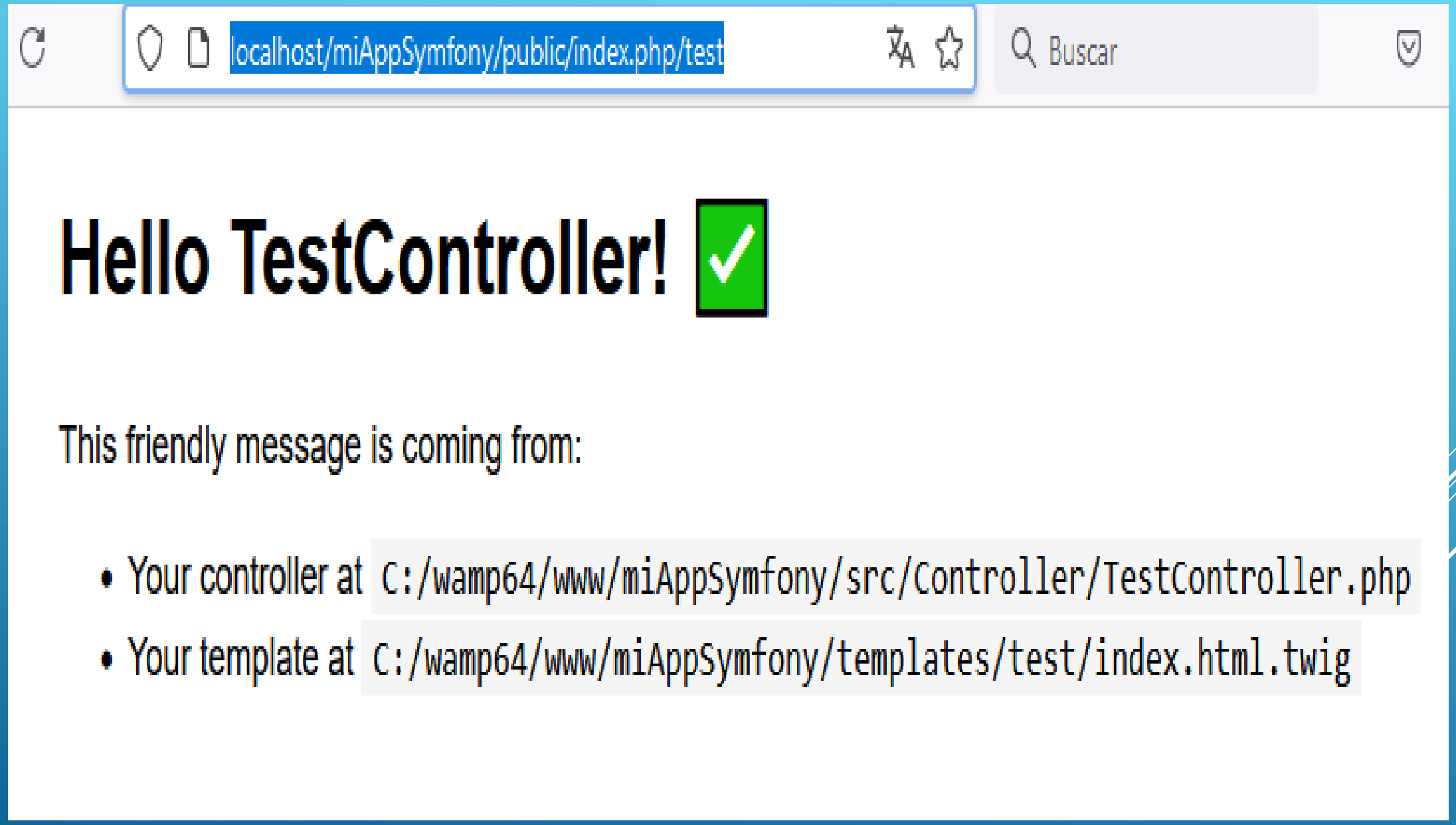
LOS CONTROLADORES

En un controlador podemos tener más de una función, lo único que tenemos que asignar justo encima de estas funciones son las rutas que los gobiernan para ser ejecutadas, por lo que vamos a añadir otra función al control, para realizar una respuesta sin necesidad de renderizar a una plantilla, y dar la respuesta desde el mismo control.

Básicamente, este primer controlador lo que está diciendo es que cada vez que se encuentre la URL /test, symfony busca entre sus controladores donde está esa ruta, y cuando la encuentra ejecuta la función que lleva asociada, que es siempre la inmediatamente debajo.

Para ver el resultado de este método:

localhost/miAppSymfony/public/index.php/test


localhost/miAppSymfony/public/index.php/test
Buscar
Hello TestController!
This friendly message is coming from:
Your controller at C:/wamp64/www/miAppSymfony/src/Controller/TestController.php
Your template at C:/wamp64/www/miAppSymfony/templates/test/index.html.twig

```
{% extends 'base.html.twig' %}

{% block title %}Hello TestController!{% endblock %}

{% block body %}
<style>
    .example-wrapper { margin: 1em auto; max-width: 800px; width: 95%; font: 18px/1.5 sans-serif; }
    .example-wrapper code { background: #F5F5F5; padding: 2px 6px; }
</style>

<div class="example-wrapper">
    <h1>Hello {{ controller_name }}! □</h1>


    This friendly message is coming from:
    <ul>
        <li>Your controller at <code>C:/wamp64/www/miAppSymfony/src/Controller/TestController.php</code></li>
        <li>Your template at <code>C:/wamp64/www/miAppSymfony/templates/test/index.html.twig</code></li>
    </ul>
</div>
{% endblock %}
```

LOS CONTROLADORES

El siguiente ejemplo de controlador es para emitir un mensaje desde dentro del control, sin llamar a ninguna plantilla.

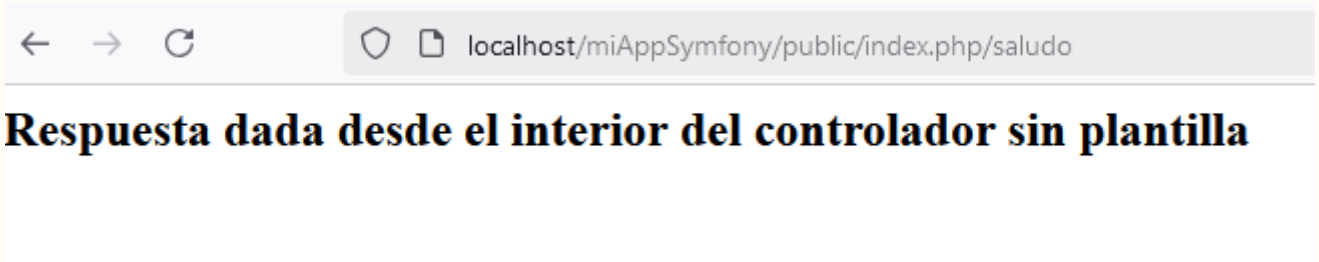
Como podemos ver se ha hecho un nuevo método, gobernado por otra url , y con un name único.

Para ver el resultado de este método:

localhost/miAppSymfony/public/index.php/saludo

wampmanager.conf | TestController.php

```php
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class TestController extends AbstractController
{
    #[Route('/test', name: 'app_test')]
    public function index(): Response
    {
        return $this->render('test/index.html.twig', [
            'controller_name' => 'TestController',
        ]);
    }


    #[Route('/saludo', name: 'app_saludo')]
    public function saludo(): Response
    {
        return new Response('<html><body><h2>Respuesta dada desde el interior del controlador sin plantilla</h2></body></html>');
    }
}
```



LAS RUTAS, SUS SINTAXIS

Las rutas en Symfony son patrones que informan a la aplicación sobre que acción dentro de cada controlador resolverá una petición determinada. También puede definirlos parámetros esperados por esa acción.

Se definen en las anotaciones de cada método, en la primera ejecución, el core de Symfony recogerá estas anotaciones y las dispondrá en la cache, para las siguientes ejecuciones.

LAS RUTAS, SUS SINTAXIS

Ruta:

✓Va entrecomillas y definiremos la parte de la url a la que hay que llamar, sin incluir el dominio.

Por ejemplo: /users o bien /blog/comments.

✓Se acepta una sintaxis que nos permite establecer una parte de la ruta como variable, y que la vamos recibir como parámetro en la acción, esto se denomina **placeholder.**

LAS RUTAS, SUS SINTAXIS

- ✓ Si definimos un placeholder, que es un nombre entre llaves en la ruta, "/blog/{placeholder}", podremos recibir en la acción una variable $placeholder con ese valor, y presentar solo la información de la página requerida en la petición.
- ✓ El placeholder en la ruta y en el requerimiento va sin el simbolo '$'.

LAS RUTAS, SUS SINTAXIS

name:

✓Es el nombre interno que podremos utilizar para identificar las rutas que definimos. Sigue el formato name="nombreRuta". **Debe ser único**

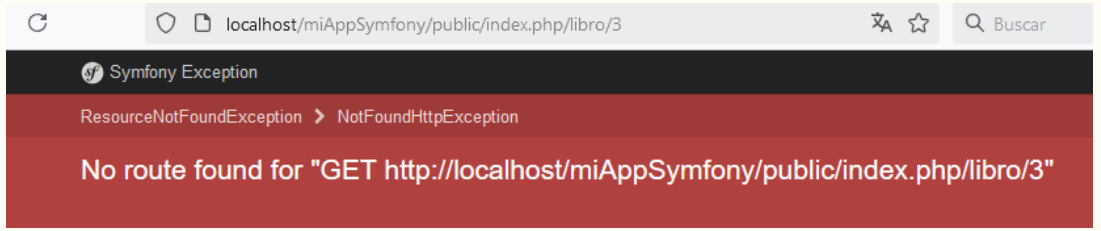
requirements:

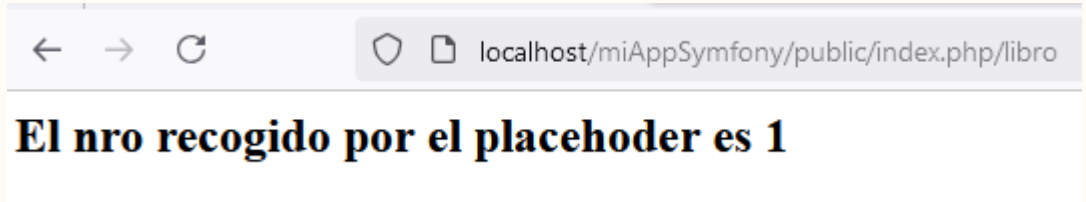
✓Es un es una **expresión regular** de requisitos **para los placeholders**.

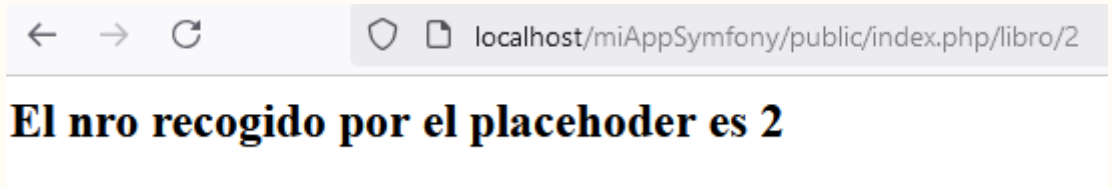
✓De esta manera se permite discriminar o encauzar rutas en función de su valor.

✓En el requisito también se puede poner que tome valores específicos, por ejemplo valores de una lista cerrada para ello se pondrá:

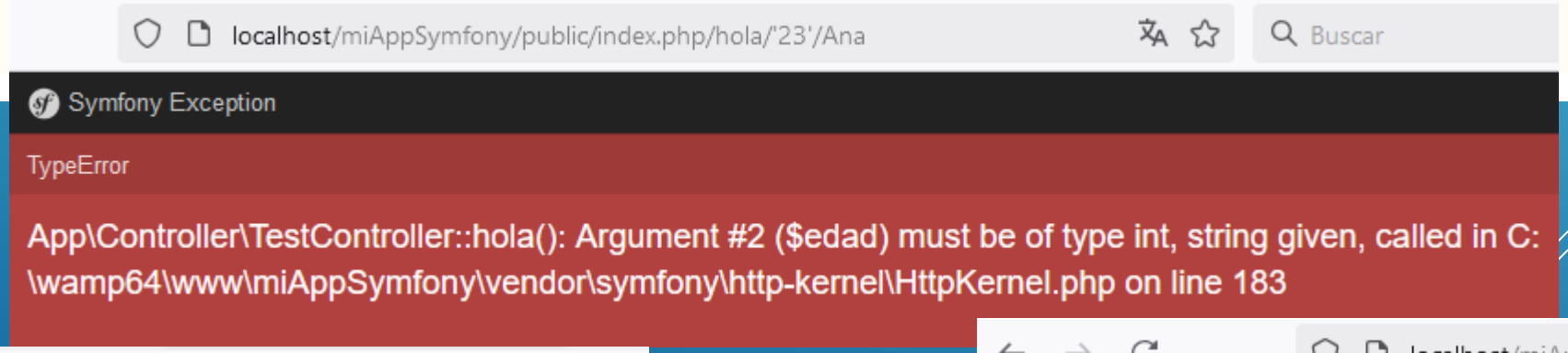
```php
<?php

namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;

class TestController extends AbstractController
{
    #[Route('/test', name: 'app_test')]
    public function index(): Response
    {
        return $this->render('test/index.html.twig', [
            'controller_name' => 'TestController',
        ]);
    }


    #[Route('/saludo', name: 'app_saludo')]
    public function saludo(): Response
    {
        return new Response('<html><body><h2>Respuesta dada desde el interior del controlador sin plantilla</h

    }

    #[Route('/libro/{pagina}', name: 'app_libro', requirements:['pagina'=>('1|2')])]
    public function libro($pagina=1): Response
    {$nro=$pagina;
        return new Response("<html><body><h2>El nro recogido por el placehoder es $nro </h2></body></html>");
    }
}
```

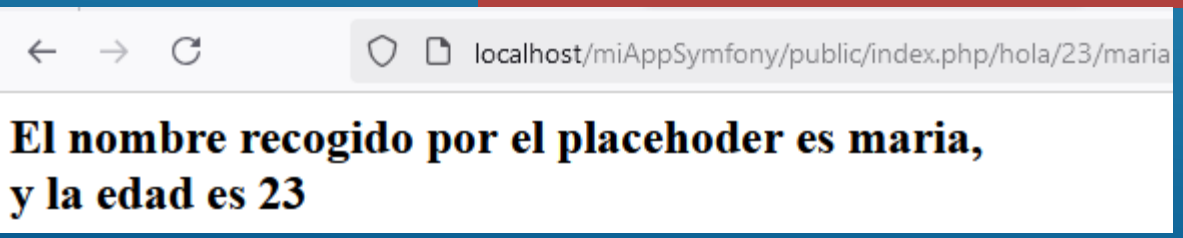






```php
    #[Route('/saludo', name: 'app_saludo')]
    public function saludo(): Response
    {
        return new Response('<html><body><h2>Respuesta dada desde el interior del controlador sin plantilla</h2></body></html>');
    }

    #[Route('/libro/{pagina}', name: 'app_libro', requirements:['pagina'=>('1|2')])]
    public function libro($pagina=1): Response
    {$nro=$pagina;
        return new Response("<html><body><h2>El nro recogido por el placehoder es $nro </h2></body></html>");
    }

    #[Route('/hola/{edad}/{nombre}', name: 'app_hola', requirements:['nombre'=>'[a-zA-Z]{2,50}'])]
    public function hola($nombre,int $edad): Response
    {$nomb=$nombre;
        return new Response("<html><body><h2>El nombre recogido por el placehoder es $nomb, y la edad es $edad </h2></body></html>");
    }
}
```





LOS VERBOS DE LAS RUTAS

Una solicitud http tiene un método de transmisión de parámetros llamados verbos.

Si se ejecuta directamente desde el navegador tiene el verbo GET, si la solicitud es llamada desde un formulario generalmente tiene el verbo POST.

En las rutas es posible especificar el método en la anotación de la ruta, que por defecto es GET.

```
#[Route('/libro/{pagina}', name: 'app_libro', requirements:['pagina'=>('1|2')] methods: [‘GET’,’POST’])]
```

Aquí estamos autorizando a que la ruta se ejecute tanto con el método POST como con el GET. Indicar el verbo de la ruta permite un mejor control de la solicitud y puede evitar algunas vulneraciones de seguridad.

PARA COMPROBAR LAS RUTAS GENERADAS EN EL PROYECTO:

Desde el cmd :

```
php bin/console debug:router
```

```
C:\wamp64\www\miAppSymfony>php bin/console debug:router
 ------------------------- -------- -------- ------ ----------------------------------
  Name                      Method   Scheme   Host   Path
 ------------------------- -------- -------- ------ ----------------------------------
  _preview_error            ANY      ANY      ANY    /_error/{code}.{_format}
  _wdt                      ANY      ANY      ANY    /_wdt/{token}
  _profiler_home            ANY      ANY      ANY    /_profiler/
  _profiler_search          ANY      ANY      ANY    /_profiler/search
  _profiler_search_bar      ANY      ANY      ANY    /_profiler/search_bar
  _profiler_phpinfo         ANY      ANY      ANY    /_profiler/phpinfo
  _profiler_xdebug          ANY      ANY      ANY    /_profiler/xdebug
  _profiler_font            ANY      ANY      ANY    /_profiler/font/{fontName}.woff2
  _profiler_search_results  ANY      ANY      ANY    /_profiler/{token}/search/results
  _profiler_open_file       ANY      ANY      ANY    /_profiler/open
  _profiler                 ANY      ANY      ANY    /_profiler/{token}
  _profiler_router          ANY      ANY      ANY    /_profiler/{token}/router
  _profiler_exception       ANY      ANY      ANY    /_profiler/{token}/exception
  _profiler_exception_css   ANY      ANY      ANY    /_profiler/{token}/exception.css
  app_aleatorio             ANY      ANY      ANY    /aleatorio/number
  app_project1              ANY      ANY      ANY    /project1
  app_project2              ANY      ANY      ANY    /project2
  app_test                  ANY      ANY      ANY    /test
  app_saludo                ANY      ANY      ANY    /saludo
  app_libro                 ANY      ANY      ANY    /libro/{pagina}
  app_hola                  ANY      ANY      ANY    /hola/{edad}/{nombre}
  areaPoligonos             ANY      ANY      ANY    /area/{poligono}/{base}/{altura}
  app_edad                  ANY      ANY      ANY    /edad/{id}
  app_aleatorio2            ANY      ANY      ANY    /aleatorio
 ------------------------- -------- -------- ------ ----------------------------------
```