Symfony 5

Curso de iniciación

¡BIENVENIDOS!

Soy David las Heras

Desarrollador de aplicaciones web

 linkedin.com/in/david-las-heras-ferris-4a2766aa/

ÍNDICE

- Conocimientos previos
- Conceptos básicos
  - Symfony
  - Patrón MVC
  - Composer
  - Flex
  - Comando Symfony
  - Tipos de proyecto

ÍNDICE

ÍNDICE

ÍNDICE

ÍNDICE

ÍNDICE

Conocimientos Previos (requisitos):

- Conocimientos básicos de HTML y CSS.
- Recomendable conocimientos básicos en framework de CSS Bootstrap 4
- Conocimientos en lenguajes de programación.
- Recomendable conocimientos en PHP ya que sino puede ser difícil entender ciertos conceptos y seguir la didáctica del curso.
- Si no tienes conocimiento en PHP o tu nivel es muy bajo este curso podría ser una buena iniciación https://openwebinars.net/cursos/curso-php-basico/

¿Por qué Symfony?

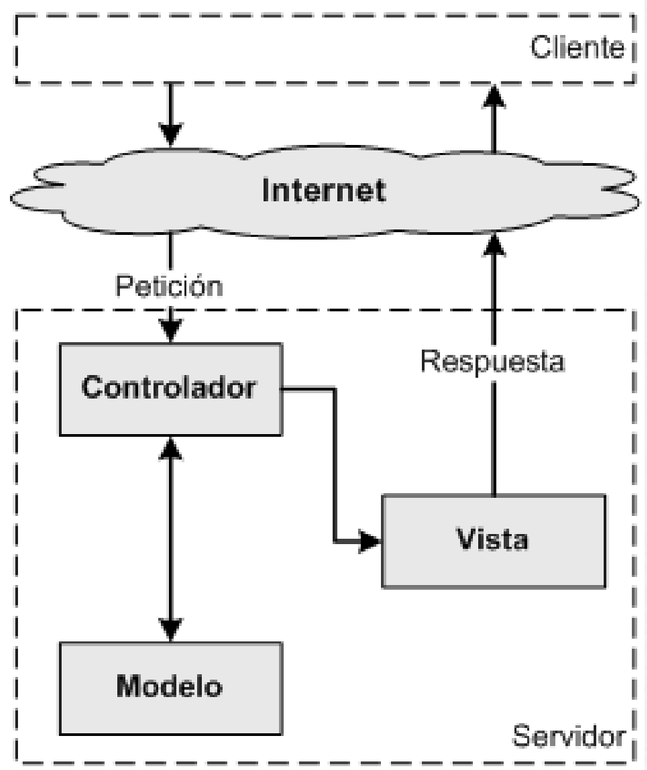
Symfony es un proyecto PHP de software libre que permite crear aplicaciones y sitios web rápidos y seguros de forma profesional

Principales características:

- Su código, y el de todos los componentes y librerías que incluye, se publican bajo la licencia MIT de **software libre**.
- La **documentación** del proyecto también es libre e incluye varios libros y decenas de tutoriales específicos.
- Los **componentes** de Symfony son tan útiles y están tan probados, que proyectos tan gigantescos como Drupal 8 están construidos con ellos.
- Basado en MVC (modelo-vista-controlador)
- …

Patrón MVC

El MVC o Modelo-Vista-Controlador es un patrón de arquitectura de software que, utilizando 3 componentes (Vistas, Modelos y Controladores) separa la lógica de la aplicación de la lógica de la vista en una aplicación. Symfony utiliza una adaptación de este patrón



Composer

Se trata de un gestor de dependencias para PHP haciendo fácil la instalación, actualización y eliminación de librerías en proyectos. Composer descargará dichas librerías en el directorio /vendor dentro del proyecto.

Proporciona un estándar para administrar, descargar e instalar dependencias.

Link de descarga:
https://getcomposer.org/download/

Flex

Flex es un plugin de composer. Cuando se instala un bundle flex ‘intercepta’ la instalación. Los bundles tienen un fichero de configuración denominado “recipe” (receta) que contiene todo lo necesario para poder integrar el bundle con la app, esta receta puede crear ficheros en nuestro proyecto, o simplemente definir nuevas variables de configuración en el fichero .env, por ejemplo el bundle utilizado por doctrine creará una variable de entorno en el fichero .env

Flex

```
> composer require doctrine/doctrine-bundle
```

```
# .env

###> doctrine/doctrine-bundle ###

DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/db_name?serverVersion=5.7

###< doctrine/doctrine-bundle ###
```

Los comentarios ###> doctrine/doctrine-bundle ### y ###< doctrine/doctrine-bundle ### son insertados por flex para poder gestionar la instalación y eliminación del propio bundle.

Comando symfony

Es un binario que se instala en el sistema para facilitar el uso de comandos symfony en el proyecto.

Principalmente lo utilizaremos para:

- Crear proyectos
- Inicializar el servidor web integrado en PHP

Link de descarga:
https://symfony.com/download

Tipos de proyecto

- Proyecto web

```
> symfony new nombre_del_proyecto --full
```

```
> composer create-project symfony/website-skeleton nombre_del_proyecto
```

- Proyecto para microservicios, aplicación de consola o api

```
> symfony new nombre_del_proyecto
```

```
> composer create-project symfony/skeleton nombre_del_proyecto
```

https://symfony.com/doc/current/setup.html#creating-symfony-applications

Estructura de directorios

proyecto/
- **assets/** Ficheros front como css, js, imágenes, etc...
- **bin/** Ejecutables para comandos de consola
- **config/** Archivos de configuración (paquetes instalados, rutas, servicios)
- **public/** Document root de nuestra app, posteriormente se encontrarán los ficheros assets
- **src/** Se encuentran todos los ficheros que forman la aplicación
  - **Command/** Comandos ejecutables a través de la consola
  - **Controller/** Controladores
  - **DataFixtures/** Datos de prueba
  - **Entity/** Entidades de base de datos
  - **EventSubscriber/**Capturadores de eventos
  - **Form/** Formularios
  - **Migrations/** Ficheros de migración usados para mantener un versionado de la base de datos
  - **Repository/** Gestionan las entidades, son los encargados de realizar consultas a la base de datos
  - **Security/** Contiene clases para el soporte de seguridad
  - **Twig/** Contiene extensiones de twig
- **templates/** Vistas en twig
- **tests/** Test de pruebas
- **translations/** Traducciones (ficheros de diccionario)
- **var/** Contiene ficheros de cache y logs
- **vendor/** Librerías de terceros (bundles)
- **.env** Este fichero contiene opciones que cambian de una máquina a otra (por ejemplo, de la máquina de desarrollo al servidor de producción) pero que no cambian el comportamiento de la aplicación. En producción serán definidas en el servidor.

Fichero configuración .env

Este fichero contiene opciones que cambian de una máquina a otra (por ejemplo, de la máquina de desarrollo al servidor de producción) pero que no cambian el comportamiento de la aplicación.

En producción podrían estar definidas a nivel de PHP, el ejemplo más claro sería APP_ENV.

Nota! El fichero .env no se modifica(será el que subamos al control de versiones), las modificaciones se harán sobre un fichero para cada entorno

- **.env.local**(no se sube al repositorio) será el que modificaremos sobrescribiendo los datos que necesitemos del .env
- **.env.test** será el que contendrá las variables de entorno para los tests y será el que subamos al repositorio de git
- **.env.test.local**(no se sube al repositorio) tendrá lo mismo que el fichero .env.test pero sobrescribiendo los valores para que funcione en nuestra máquina

¿Qué es un controlador?

Será el encargado de gestionar la petición realizada y devolver una respuesta.

La capa de negocio se encontrará en los servicios y la capa de persistencia estará relacionada con la entidades y repositorios.

¿Qué es twig?

Es un gestor de plantillas para PHP, el cual permite generar archivos de texto que puede arrojar resultados en formatos como HTML, XML, CSV…

Los motivos principales para su uso son:

- Limpio: Utiliza una sintaxis visualmente muy limpia y concisa
- Rápido: Compila los ficheros a código PHP
- Dispone de herencia entre plantillas: Ayuda a la flexibilidad y reutilización de código
- Dispone de filtros: Ayudan a un código mas limpio
- Fácil aprender

https://twig.symfony.com/doc/3.x/

¿Qué es doctrine?

Es el ORM utilizado por symfony, Object-Relational Mapping, es decir, el mapeo relacional de objetos. Esto significa que va a trasladar los datos de una base de datos relacional, como puede ser MySQL o SQL Server, a un sistema de clases y de objetos, donde las clases serían las tablas y los registros pasarían a ser lo equivalente a objetos.

Más información sobre el uso de doctrine:
https://symfony.com/doc/current/doctrine.html

¿Qué es una entidad?

Es la clase equivalente a la tabla de base de datos

Más información sobre entidades:
https://symfony.com/doc/current/doctrine.html#creating-an-entity-class

¿Qué es un repositorio?

Contiene todos los métodos de consultas asociadas a una entidad, por defecto se pueden encontrar los siguiente métodos:

- find($id)
- findAll
- findOneBy(array de criterios)
- findBy{propiedad}
- findOneBy {propiedad}

Además podremos crear nuestros propios métodos de búsquedas

https://symfony.com/doc/current/doctrine.html#querying-for-objects-the-repository

¿Qué es un servicio?

Son clases utilizadas para aislar y reutilizar una funcionalidad, estas clases pueden ser invocada en cualquier lugar de la aplicación gracias a la gestión de dependencias

¿Qué son los bundles?

Son un conjunto de archivos propios o de terceros que implementan una funcionalidad, por ejemplo, el bundle de doctrine que nos permite utilizar doctrine como ORM.

https://symfony.com/doc/current/bundles/DoctrineBundle/index.html

¿Qué son los comandos?

Symfony permite la ejecución de comandos a través de su bundle console.

```
> composer require symfony/console
```

Permitiendo que otra series de bundle puedan apoyarse en este para realizar ciertas tareas, como puede ser la generación de código.

Por ejemplo el ‘MakerBundle’ es un bundle usado para la generación de código automática, su uso es recomendado por parte del equipo de Symfony

Generar controlador de ejemplo

```
> php bin/console make:controller clientes
```

https://symfony.com/doc/current/bundles/SymfonyMakerBundle/index.html

¿Qué son los mensajes flash?

Es un componente que proporcionan una forma de configurar y recuperar mensajes por sesión.

El flujo de trabajo habitual sería configurar mensajes flash en una solicitud y mostrarlos después de una redirección de página. Por ejemplo, un usuario envía un formulario que llega a un controlador de actualización y, después de procesar, el controlador redirige la página a la página actualizada o a una página de error. Los mensajes flash configurados en la solicitud de página anterior se mostrarían inmediatamente en la carga de la página siguiente para esa sesión.

https://symfony.com/doc/current/controller.html#flash-messages

Otros mensajes flash:
https://symfony.com/doc/current/components/http_foundation/sessions.html#flash-messages

Excepciones

Cuando la aplicación no es capaz de continuar su flujo de trabajo lanzará o lanzaremos una excepción. Por defecto las excepciones mostrarán mucha información en el entorno de dev mientras que en prod esta información se reduce debido a la sensibilidad de la información mostrada, se mostrará un twig.

Podemos reproducir el template de dichas plantillas en el entorno de dev de la siguiente manera:
https://dominio/_error/{codigo_error}

Por ejemplo
https://dominio/_error/404

Más información sobre el testeo de las páginas de error:
https://symfony.com/doc/current/controller/error_pages.html#testing-error-pages-during-development

Personalizar diseño excepciones(twig)

Es posible modificar los mensajes de error de twig sobre escribiendo las plantillas predefinidas.

Estas plantillas estarán alojadas en el directorio **templates/bundles/TwigBundle/**

Crearemos un twig para la personalización de cada página de error:

- **error.html.twig:** vista que será mostrada por defecto
- **404error.html.twig:** vista que será mostrada en caso de error 404

Más información acerca de la personalización de páginas de error:
https://symfony.com/doc/current/controller/error_pages.html#overriding-the-default-error-templates

Componente validador

La validación es una tarea muy común en las aplicaciones web. Los datos ingresados en los formularios deben ser validados. Los datos también deben validarse antes de que se escriban en una base de datos o se pasen a un servicio web.

Symfony proporciona un componente Validator que hace que esta tarea sea fácil y transparente.

Componente validador:

https://symfony.com/doc/current/validation.html

Listado de validaciones existentes:
https://symfony.com/doc/current/reference/constraints.html

Validaciones personalizadas

Symfony permite crear validaciones personalizadas.

Para esta tarea nos vamos a apoyar en makerBundle ya que nos provee un comando para hacernos mas sencilla esta tarea, nos creará los ficheros necesarios con la estructura básica:

```
> php bin/console make:validator
```

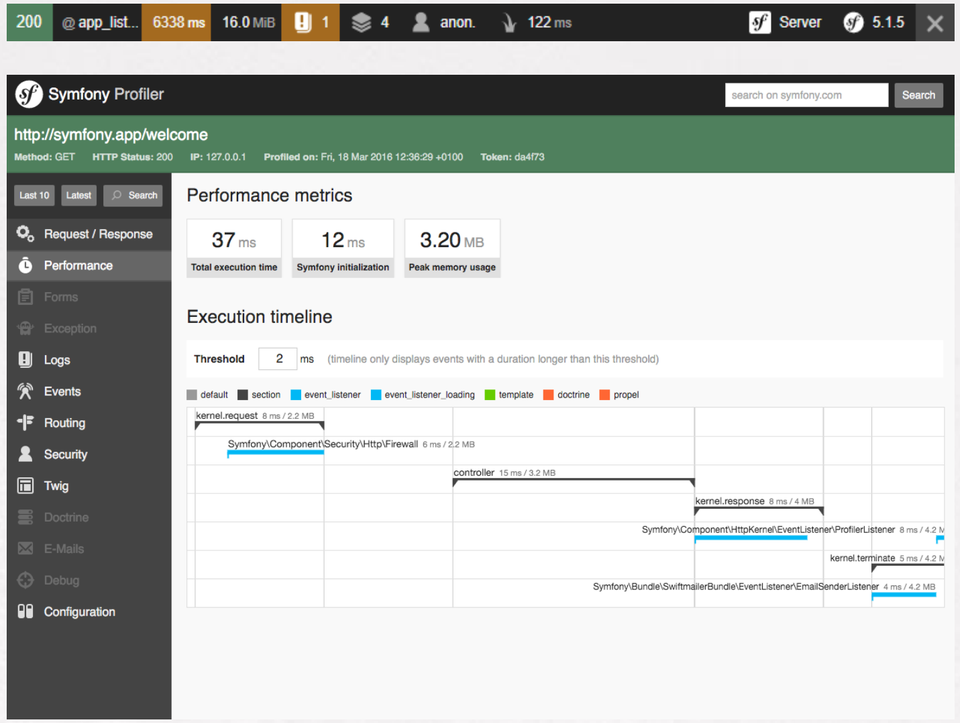
Más información:
https://symfony.com/doc/current/validation/custom_constraint.html

Depuración

Podemos encontrar varias herramientas para la depuración: Toolbar, profiler y función dump

Toolbar y profiler: una herramienta que nos provee de información en cada request, rendimiento, logs, eventos, enrutamiento…

https://symfony.com/doc/current/profiler.html



Depuración

- **Función dump:** Esta función nos muestra la información que contiene una variable, digamos que es como un var_dump pero enriquezido.

  - Se puede utilizar en php:
    `dump($var)`

  - En twig se presenta de dos formas:

    - Realizando un echo del contenido
      `{{ dump var }}`

    - Almacenando la variable y añadiéndola a la toolbar
      `{% dump var %}`

https://symfony.com/doc/current/components/var_dumper.html

Servidor web

Php proporciona un servidor web. Symfony utiliza dicho servidor web, el cual nos permite ejecutar proyectos web sin necesidad de recurrir a aplicaciones de terceros como podría ser xampp, wampserver, etc...

- Arranca servidor integrado:

```
> php bin/console server:start
```

```
> symfony server:start
```

- Arranca servidor integrado en background (segundo plano):

```
> php bin/console server:start --d
```

```
> symfony server:start --d
```

https://symfony.com/doc/current/setup.html#running-symfony-applications

MakerBundle

Es un bundle utilizado para la generación de código, recomendado por symfony y usado durante toda su documentación como base en el desarrollo

https://symfony.com/doc/current/bundles/SymfonyMakerBundle/index.html

Enrutamiento y controladores

Enrutamiento:

https://symfony.com/doc/current/routing.html

Controladores:

https://symfony.com/doc/current/controller.html

Vistas

https://symfony.com/doc/current/templates.html

Conceptos básicos de twig

https://twig.symfony.com/

Herencia:

https://twig.symfony.com/doc/3.x/tags/extends.html

Resumen general(estructuras de control, operadores, filtros, etc…):

https://twig.symfony.com/doc/3.x/templates.html

Listado de filtros:

https://twig.symfony.com/doc/3.x/filters/index.html

Habilidades obtenidas

- Conocimientos generales sobre arquitectura de Symfony(MVC)
- Conocimientos generales que engloban las aplicaciones en symfony
- Inyección de dependencias en Symfony
- Conocimientos básicos en enrutamiento y controladores
- Conocimientos básicos en twig
- Conocimientos básicos en doctrine

Guía de estilos y tips

- Nombre de proyectos con barra baja(_) como separación, ej, gestor_de_oficinas
- Nombre de servicios lo mas corto posible, sin que se pierda valor
- Utilizar constantes si los valores de configuración rara vez cambian, por ejemplo, el número de elementos mostrados por página
- Utilizar ParamConverters siempre que sea posible, si es una consulta compleja se utilizarán repositorios
- Anotaciones para enrutamiento
- Por regla general las rutas se definirán con un name interno separado por _ y con las rutas separadas por -, por ejemplo:

```
/**
 * @Route("/pagina-principal", name="app_index")
 */
```

Guía de estilos y tips

- Por precaución es recomendable añadir app_ a los names de las rutas, eso es debido a que otros bundles pueden tener su propio enrutamiento y es una medida de seguridad para no solaparse, ya que estos name deben ser únicos.
- Anotaciones para mapeo de las entidades
- Tipado de variables siempre que sea posible
- Tipado de los valores de retorno siempre que sea posible
- Comentar las funciones y variables siempre y cuando no quede claro tan solo con el nombre de la función, por ejemplo, la función getNombre no es necesario comentarla ya que esta claro cual es su función. Sin embargo si que es necesario indicar su valor de retorno.

Guía de estilos y tips

- Usar la estructura de directorios por defecto, esto no quiere decir que no podamos crear nuestros propios directorios si fuera necesario, pero no modificar las existentes y reinventar la rueda
- Utilice una sola acción para renderizar y procesar el formulario, por ejemplo, utilizar crearTarea en lugar de mostrarFormularioTarea y guardarFormularioTarea
- Añadir , al final de los arrays, ej, [‘hola’,]
- Las {} se abren en siguiente línea tanto en definición de clases como métodos, y en la misma línea en condicionales y bucles
- Utilizar comillas simples en twig, ej, {% include ‘comunes/_menu.html.twig’ %}

Guía de estilos y tips

- Utilizar nombre con _ para separar variables twig, ej, listado_usuarios
- Ordenar los use colocando primero las clases de la app y después el resto
- Colocar primero los servicios de la app en la inyección de dependencias
- Utilizar interfaces en los servicios de symfony
- Utilizar entity manager en los controladores de la siguiente manera:
  `$em = $this->getDoctrine()->getManager();`
- Utilizar entity manager en los controladores de la siguiente manera:
  `EntityManagerInterface $em`
- En las condiciones colocar primero el valor y luego la variable, ej, null === $var, si hacemos esto nunca tendremos problemas de asignación, $var = null

Guía de estilos y tips

- En los condiciones de una sola línea no indicar las {}
- No tabular los arrays, se trata de una practica antigua, en la cual se pierde tiempo y se pierde la posibilidad de formatear el código automáticamente con el ID utilizado
- Los programadores tienen que programar, no son redactores. El código sufre modificaciones con el tiempo y los comentarios pierden su valor y a veces hasta son contradictorios, por este motivo “Comenta lo necesario y coméntalo bien”
- Comentar los trozos de código que sean estrictamente necesario por alguna peculiaridad, ej, validamos si el valor es -3 debido a una mala conexión con el webservice de amazon si es el último martes del mes debido a un conteo de stock por parte de su plataforma. No comentar cosas estilo if (0 === $numUsuarios)// no hay usuarios

Guía de estilos y tips

- Utilizar ‘’ cuando el string no contenga variables. Ej, $saludo = ‘hola’;
- Utilizar “” cuando el string contenga variables. Ej, $despedidaPersonalizada = “Adiós $nombreUsuario”
- No concatenar siempre y cuando sea posible. Ej, Usar “Hola $nombreUsuario” en lugar de ‘Hola ’ . $nombreUsuario
- Los parámetros definidos en config/services.yaml deben estar separados por _ e intentar que sean una o dos palabras

Notas ficheros

1. **config/routes.yaml:** Configuración de rutas, por regla general el enrutamiento se hará a nivel de controlador con anotaciones(es lo recomendado por symfony)
2. **.env:** Contiene las variables que cambian según la máquina, como puede ser el entorno, el acceso a la bbdd, etc…
3. **.env.local:** Este fichero será el utilizado para sobreescribir los valores de .env. Cada persona del equipo de desarrollo tendrá el suyo, al igual que en producción(o definir todas las variables en el server)

Comandos

1. **symfony new curso_principiante –full:** Crear proyecto web
2. **symfony server:start:** arrancar servidor web
3. **php bin/console doctrine:database:create:** Crear la bbdd que se ha definido en parámetro de configuración por entorno
4. **php bin/console make:controller:** {nombreControlador}: Crear controlador y su vista
5. **php bin/console debug:router:** Mostrar todas las rutas definidas
6. **php bin/console cache:clear:** Eliminar caché generada por symfony, tanto el enrutamiento como las plantillas twig son compilados(entre otros ficheros), por este motivo hay veces que en dev un nuevo enrutamiento o modificación de twig no funcione. Para solucionar este problema se puede utilizar el comando o en su defecto eliminar la carpeta de entorno encontrada dentro de var/cache. En prod siempre habrá que limpiar dicha cache con cada nueva subida