Florida Universitària

Desarrollo Web en Entorno Servidor

AEV1
Creación de un MVC

Ciclo Formativo de Desarrollo de Aplicaciones Web

Realizado por: Ana Piqueras Jiménez

Índice:

1. Introducción de la actividad

1.1. Crear un Modelo Vista Controlador.

Esta actividad evaluable consiste en crear una aplicación que cumpla los criterios vistos en el bloque 1.
Debemos crear un proyecto que cumpla con el patrón MVC, que tenga un sistema de enrutamiento mediante el autoload de Composer y los namespaces. También usaremos TWIG para implementar las plantillas visuales.

Para ello se pide:

1. Buscar una plantilla gratuita de HTML en internet con la que debemos implementar TWIG y la herencia de este. Podéis usar las plantillas, por ejemplo, que están en la siguiente web gratuita https://all-free-download.com/free-website-templates/ o en esta otra https://templatemo.com/ lo único que no deben ser spa.

2. Crea un proyecto con la estructura típica que se ha visto en el tema 1 y 2 de MVC y TWIG.

3. Instala Composer (el gestor de dependencias que vamos a usar todo el curso) e implantaremos las especificaciones de autoload PSR-4.

4. Es importante que durante el desarrollo del proyecto se use un repositorio GIT, en este caso usaremos GitHub donde se os asignará a cada uno un repositorio donde deberéis hacer la subida de toda la actividad evaluable.

5. La estructura de carpetas estará compuesta por:

• **Config** → Obtén la información de conexión de BB.DD. desde un fichero JSON que se conectará a la BB.DD. de MySQL que se entrega con la documentación de la actividad. (empresa.sql).
Además, se incorporará otro fichero JSON con el enrutado de la aplicación. Es decir, que rutas se corresponden con qué controladores. Crea las clases necesarias para la gestión de esta información.

• **Public** → Donde deberá estar toda la lógica del HTML publica, CSS, JS, etc… de la plantilla descargada. El archivo index.php deberá contener la información necesaria para acceder al autoload.

• **Src** → Contendrá la lógica de todo el proyecto mediante las siguientes subcarpetas:

- **Controllers**→ Estará contenida toda la lógica de los controladores que sea necesaria para hacer funcionar la aplicación con todas las rutas permitidas.
- **Models**→ Se adjunta un fichero SQL para exportar la BB.DD., empresa.sql. Con la

que poder importarla a tú MySQL y generar las clases para obtener los datos necesarios. Toda la lógica del modelado de la BB.DD. debe estar en esta carpeta. Importante recordar que cada tabla deberá estar contenida en una clase diferente.

- **Core** → En esta carpeta incluiremos todas las clases necesarias para el autoload, interfaces y demás clases que harán funcionar el MVC.
  - **Interfaces** → Deberá contener todos los interfaces que sean necesarios para definir los métodos mínimos que garanticen que la aplicación pueda ser escalada adecuadamente como un MVC.
- **Templates**→ Donde tendremos contenidas todas las plantillas TWIG de nuestra aplicación. A partir de la plantilla HTML descargada deberá extenderse las diferentes plantillas TWIG para que contenga la información requerida en las diferentes páginas del ejercicio.

6. La aplicación solo contendrá como máxima las siguientes rutas:

6.1. Usaremos siempre TWIG para toda la lógica de la vista.

6.2. Todas las vistas han de tener un link o botón, según sea más adecuado por la plantilla descargada, para poder volver a la ruta anterior.

6.3.**/ o raíz** → En esta vista mostraremos la pantalla de inicio como mínimo una imagen, que deberá cambiar cada vez que recarguemos la página. Deberá haber como mínimo 3 imágenes diferentes para alternarse y deberán estar contenidas con la plantilla HTML descargada previamente. Además, añadiremos según la estructura de la plantilla unos botones, links u otra acción que nos permita ir a las otras rutas que tenemos que se piden en el ejercicio.

6.4. **/listaProductos** → Cargará la lista de todos los productos. Modifica la plantilla usada en la ruta raíz y extiende de ella de una forma adecuada. La lista de los productos estará contenida en una tabla que deberá quedar bien con la plantilla que hayas descargado.

6.5. **/listaClientes** → Cargará la lista de todos los clientes.

6.6. **/detalleCliente** → Cargará los datos de un cliente. Para ello, deberemos pasar el id del cliente como parámetro de ruta. Los datos del cliente deberán mostrarse en una tabla con una estructura diferente a la tabla de lista.

6.7. **/departamentos** → Cargará una tabla con todos los departamentos y desde cada departamento podremos ir a la ruta /empleados que pertenezcan a ese departamento.

6.8. **/empleados** → Cargará una tabla con todos los empleados de la empresa si no se ha recibido parámetro correspondiente al departamento. Si recibe el id del departamento mostrará únicamente los empleados de ese departamento.

6.9. **/pedidos** → Mostrará un desplegable con el nombre de todos los clientes y una vez seleccionado el cliente correspondiente podremos pulsar un botón que nos debe mostrar en la misma página todos los pedidos de ese cliente. Pudiendo volver a seleccionar otro cliente en el desplegable. En cada pedido tendremos un link sobre el número de pedido que nos permita ir a otra plantilla /detallePedido con los detalles del pedido.

6.10. **Cualquier otra ruta** deberá mostrar una pantalla de error muy similar a la / donde además de contener todo lo que tiene que mostrar esa ruta debe mostrar un mensaje de ruta no disponible.

7. **Formato TWIG** → Hay que formatear:

7.1. Las **fechas** al formato dd/mm/aaaa usando los **filtros TWIG**.

7.2. Los **textos** deben estar todos en minúsculas, salvo la primera letra de títulos y nombres que estará en mayúsculas, para hacer esto debemos usar **filtros TWIG**, no lógica de PHP.

7.3. Todos los valores económicos de dinero deberán estar filtrados con **el símbolo de € y deberán mostrar siempre dos decimales**.

1.2. Recursos Entregados.

Presentaciones y videos de Temas 1:

- Introducción a MVC
- Espacios de nombres PHP.
- Estructura típica de un proyecto MVC (composer)
- Singleton

Presentaciones y videos de Temas 2:

- Introducción a TWIG

Material de apoyo:

- POO y MVC en PHP
- The TWIG book
- Y demás links del tema en Florida Oberta.

Recursos adicionales:

- Usar script creación BB.DD. empresa.sql en formato SQL que se ha entregado junto con los recursos de la actividad.

- Hoja de rubrica para rellenar la autorubrica y conocer los diferentes niveles de evaluación de la actividad.

1.3. Objetivos. Rúbrica.

Existen cuatro niveles de corrección para esta actividad:

- Mínimos (hasta 5 puntos)
  - Crear estructura de carpetas según el patrón MVC correctamente como se ha visto en los últimos ejemplos.
  - Realizar memoria según documentación subida en Florida Oberta.
  - Realizar video explicativo de la actividad.
  - El sistema de autoload debe cumplir el protocolo PSR-4 y estar implementado mediante el gestor de dependencias Composer.
  - La aplicación debe conectar con la BB.DD.
  - Que la ruta raíz, "/", se muestre al iniciar el proyecto desde la carpeta public, ejecutando los requisitos

- Suficiencia
  - Cumplir todos los requisitos del punto mínimo.
  - Añadir la implementación de las rutas: **"/listaProductos", "/listaClientes**" y **"/detalleCliente**".

- Notoriedad
  - Cumplir todos los requisitos del punto suficiencia.
  - Añadir la implementación de las rutas: **"/departamentos**" y **"/empleados**".
  - Todas las plantillas implementadas hasta ese momento deben poder volver a la pantalla anterior.

- Excelencia
  - Cumplir todos los requisitos del punto notoriedad.
  - Añadir la implementación de las rutas: **"/pedidos**" y **"/detallePedido**".

2. Solución de la actividad.

Esta actividad corresponde a una actividad evaluable, que me permite afianzar conceptos básicos del MVC para PHP con un sistema de enrutado mediante autoload de Composer, los namespaces y la implementación de las plantillas visuales mediante TWIG.

2.1. Descripción de la solución y porqués.

Esta solución implementa mediante un archivo JSON las estructuras que voy a gestionar en mi aplicación y a las que redirigiré si no las reconoce. He usado otro archivo JSON para

obtener los datos de configuración de la BBDD que voy a usar y otro para crear la configuración del Composer.
He creado un patrón singleton para acceder a la BBDD.
La estructura de la aplicación se basa en una carpeta para la configuración: *config,* otra de acceso público: *public,*y otra llamada *src* donde está ubicada toda la lógica del MVC y el core del mismo.
También he hecho uso de TWIG para implementar las plantillas HTML.

2.1.1. Porqué has implementado esta solución.

Esta solución la he implementado porque es lo que se pedía el enunciado.
Con ella se garantiza en todo momento que el cliente ve únicamente lo que yo quiero que vea.
He establecido por separado la lógica para los controladores, los modelos y las vistas siguiendo un patrón MVC, he implementado las especificaciones de autoload PSR-4 y la plantilla HTML que me he descargado de internet con TWIG.

2.1.2. Y por qué consideras que es la solución adecuada.

Considero que es la solución más adecuada según los conocimientos que tengo hasta ahora, ya que mantiene separados los datos de la lógica, así como el *core* de la propia aplicación. He intentado reutilizar lo máximo el código, he establecido como patrón de diseño un MVC y he implementado TWIG. Siendo una solución completa y extensible según las necesidades que tengamos.

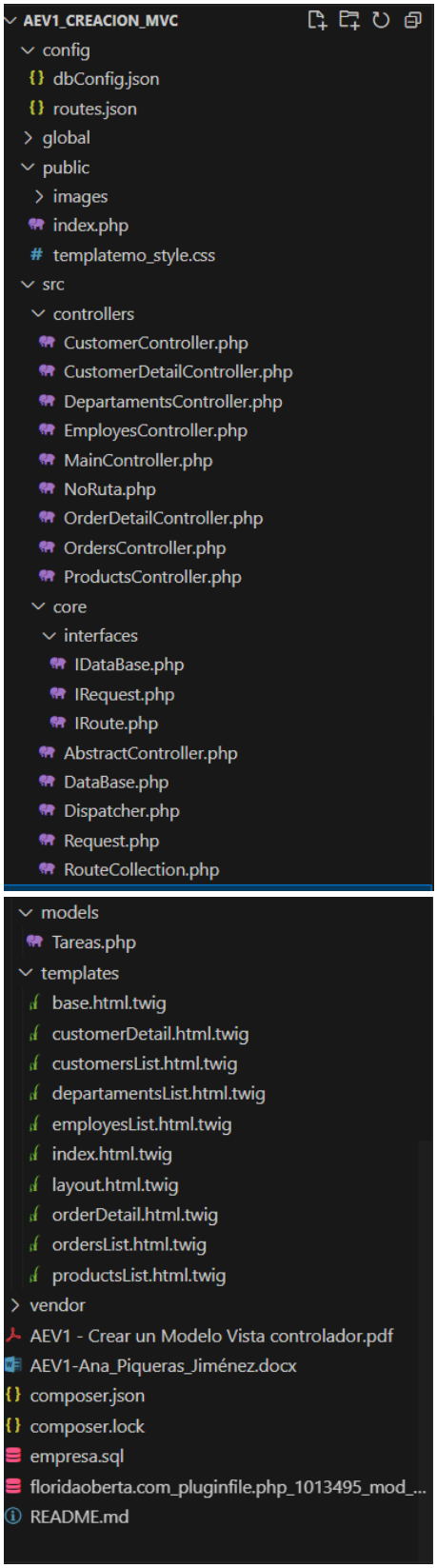
2.1.3. Y por qué otra solución no es más adecuada.

Con los conocimientos que tengo hasta ahora, no encuentro una solución más adecuada

2.2. Repositorio de Git de la actividad.

2.3. Árbol y estructura del proyecto.

La estructura del proyecto para esta actividad es:



2.4. Auto-Rúbrica

Nivel de Rúbrica	Superado	No superado	Observaciones
Mínimos	X		
Suficiencia	X		
Notoriedad	X		
Excelencia	X		Tuve problemas para formatear al euro, pero lo solucioné. También tuve un problema para recoger el valor que venía por post, pero era por un error en la etiqueta html form