

Desarrollo Web en Entorno Servidor

AEV1

**Modelo Vista**

**Controlador con Routing**

Ciclo Formativo de Desarrollo de Aplicaciones Web

Realizado por: GEORGI NEDYALKOV VANKOV

Índice:

[1. Introducción de la actividad 2](#_Toc145525805)

[1.1. Enunciado. 2](#_Toc145525806)

[1.2. Recursos Entregados. 4](#_Toc145525807)

[1.3. Objetivos. Rúbrica. 4](#_Toc145525808)

[2. Solución de la actividad. 5](#_Toc145525809)

[2.1. Descripción de la solución y porques. 5](#_Toc145525810)

[2.1.1. Porqué has implementado esta solución. 5](#_Toc145525811)

[2.1.2. Y por qué consideras que es la solución adecuada. 5](#_Toc145525812)

[2.1.3. Y por qué otra solución no es más adecuada. 6](#_Toc145525813)

[2.2. Repositorio de Git de la actividad. 6](#_Toc145525814)

[2.3. Árbol y estructura del proyecto. 6](#_Toc145525815)

[2.4. Auto-Rúbrica 8](#_Toc145525816)

# Introducción de la actividad

## Enunciado.

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:
6. **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.
  1. **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.
  2. **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.*
  3. **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.
  4. **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.
  5. Usaremos siempre TWIG para toda la lógica de la vista.
  6. 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.
  7. **/ 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.
  8. **/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.
  9. **/listaClientes** → Cargará la lista de todos los clientes.
  10. **/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.
  11. **/departamentos** → Cargará una tabla con todos los departamentos y desde cada departamento podremos ir a la ruta **/empleados** que pertenezcan a ese departamento.
  12. **/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.
  13. **/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.
  14. **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.
  15. Las fechas al formato dd/mm/aaaa usando **los filtros TWIG**.
  16. 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**.
  17. Todos los valores económicos de dinero deberán estar filtrados con **el símbolo de € y deberán mostrar siempre dos decimales.**
  18. La aplicación solo contendrá como máxima las siguientes rutas:
  19. **Formato TWIG** → Hay que formatear:
  20. Realizar memoria con el formato que está en Florida Oberta, rellenando todos los puntos que indica.
  21. Realizar video explicando cómo funciona la aplicación desarrollada de entre 5 y 15 minutos. El video mostrará como se ejecuta la aplicación, debéis salir con la cámara activa en alguna esquina del video durante todo el video. Al igual que la explicación debéis darla vosotros, intentando evitar leer un documento.
  22. Entregar Autorubrica con vuestra propia evaluación, usando la rúbrica entregada.

## Recursos Entregados.

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.

Entregar:

* Fichero ZIP con la estructura completa del proyecto, incluyendo todos sus archivos. No es necesario incluir la carpeta vendor. ***Este archivo a de ser exactamente igual a la versión subida en el repositorio de Git***.
* Incluir ruta con el link de OneDrive donde habéis subido el video y aseguraros que esta compartido conmigo y puedo acceder.
* Incluir archivo en formato Word o similar incluyendo la portada.

## Objetivos. Rúbrica.

**Rúbrica:**

Existen cuatro niveles de corrección para esta actividad:

* + **Mínimos (hasta 5 puntos)**
  1. Crear estructura de carpetas según el patrón MVC correctamente como se ha visto en los últimos ejemplos.
  2. Realizar memoria según documentación subida en Florida Oberta.
  3. Realizar video explicativo de la actividad.
  4. El sistema de autoload debe cumplir el protocolo PSR-4 y estar implementado mediante el gestor de dependencias Composer.
  5. La aplicación debe conectar con la BB.DD.
  6. Que la ruta raíz, “/”, se muestre al iniciar el proyecto desde la carpeta public, ejecutando los requisitos
  + **Suficiencia**
  1. Cumplir todos los requisitos del punto mínimo.
  2. Añadir la implementación de las rutas: **“/listaProductos”, “/listaClientes” y “/detalleCliente”.**
  + **Notoriedad**
  1. Cumplir todos los requisitos del punto suficiencia.
  2. Añadir la implementación de las rutas: **“/departamentos” y “/empleados”**.
  3. Todas las plantillas implementadas hasta ese momento deben poder volver a la pantalla anterior.
  + **Excelencia**
  1. Cumplir todos los requisitos del punto notoriedad.
  2. Añadir la implementación de las rutas: **“/pedidos” y “/detallePedido”.**

# Solución de la actividad.

Esta actividad corresponde a la actividad evaluable 1.

Analizando la base de datos se puede deducir o hacer el símil de que se trata de una empresa de envíos, una empresa de mensajería o del sector de la logística.

Para ello hemos escogido una plantilla con unos colores simples que puedan hacerle al usuario la navegación más sencilla para poder moverse por la app, siempre según sus necesidades.

Hemos aplicado un patrón MVC y un routing, especificando las rutas de forma clara y concisa, añadiendo también una página de error o de que la ruta no existe en el caso de que si se intenta acceder a una ruta no especificada válgame la redundancia. Esta última página siempre con la opción de poder volver al menú principal para evitar atascos en la aplicación.

Todas las demás rutas tienen la capacidad de ir a cualquier otro menú sin importar en que contexto se encuentren.

# Descripción de la solución y porques.

Este proyecto implementa una arquitectura MVC (modelo-vista-controlador) mediante la cual cada clase y cada script tienen que hacer una única cosa, ya sea hacer peticiones a las bases de datos o ya sea enviando datos al modelo.

Tenemos un directorio config donde se encuentran los archivos JSON uno para la conexión a la base de datos y otro donde especificamos las rutas.

La conexión a la base de datos es una conexión con arquitectura singleton.

Tenemos otro directorio con el script de la base de datos para poder importarlo.

El directorio public se alojarán todas las carpetas disponibles para el público y que se podrán mostrar como imágenes, el índice de la aplicación y otros scripts adicionales para el front-end.

El directorio “motor” de nuestra aplicación se encuentra en la carpeta src y sus subdirectorios, cada uno de ellos dedicado solo a alojar archivos de la misma naturaleza o funciones.

Las soluciones que se han aplicado han sido según siempre los requisitos de nuestro cliente, siguiendo al detalle las rutas específicas demandadas.

# Porqué has implementado esta solución.

Una de las soluciones que hemos aplicado ha sido usar una estructura singleton para la conexión a la base de datos, al ser una aplicación relativamente pequeña en este contexto y no manejar cantidades grandes de datos.

También hemos aplicado una estructura MVC, ya que en un futuro si se desean ampliar las funcionalidades permitirá una escalabilidad más sencilla de implementar.

# Y por qué consideras que es la solución adecuada.

Considero que la solución más óptima a la hora de reutilizar código, implementar nuevas funcionalidades y de la forma que se manejan los datos.

También si esta aplicación necesita alguna solución y tiene que ser otro desarrollador el que la tiene que aplicar, podrá reutilizar código, ya que cada script mantiene su funcionalidad intentando respetar siempre al máximo los principios SOLID.

# Y por qué otra solución no es más adecuada.

De hecho, en este caso la solución más óptima respecto a la conexión a la base de datos singleton que hemos implementado, sin embargo, si se empiezan a manejar grandes cantidades de datos, no lo sería.

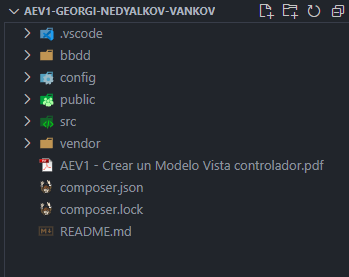
En ese momento habría que hacer un estudio y replantearse a otro tipo de arquitectura para la conexión.

# Repositorio de Git de la actividad.

# <https://github.com/magnumcrypto/AEV1-GEORGI-NEDYALKOV-VANKOV>

# Árbol y estructura del proyecto.

La estructura del proyecto para esta actividad es:

Texto

Descripción generada automáticamente

# Auto-Rúbrica

En todas las actividades evaluables al final del todo en la sección de Rúbrica se encuentra una tabla equivalente a la que se pone en este ejemplo. Es rellenarla con los criterios que vosotros pensáis que habéis hecho o no y si consideráis que hace falta indicar alguna observación que debo tener en cuenta a la hora de corregir debéis ponerla aquí.

|  |  |  |  |
| --- | --- | --- | --- |
| **Nivel de Rúbrica** | **Superado** | **No superado** | **Observaciones** |
| **Mínimos** | SI |  | Considero que los objetivos mínimos están cumplidos y se han seguido las indicaciones necesarias. |
| **Suficiencia** | SI |  | Todos los puntos están realizados y completados. |
| **Notoriedad** | SI |  | La aplicación es navegable desde cualquier punto y contexto que estes. Añadiendo todas las plantillas necesarias, rutas y demás archivos para el correcto funcionamiento. |
| **Excelencia** | SI, a medias |  | Se han implementado las rutas que se piden, pero para la excelencia considero que también es necesario buscar la eficiencia y simplicidad del código cuando sea necesario.  Se han intentado seguir los principios SOLID. |