

Desarrollo Web en Entorno Servidor

AEV1

**MVC - TWIG**

Ciclo Formativo de Desarrollo de Aplicaciones Web

Realizado por: Héctor González Mora

Índice:

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

[1.1. Enunciado. 2](#_Toc145525806)

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

[1.3. Objetivos. Rúbrica. 5](#_Toc145525808)

[2. Solución de la actividad. 6](#_Toc145525809)

[2.1. Descripción de la solución y porques. 6](#_Toc145525810)

[2.1.1. Porqué has implementado esta solución. 6](#_Toc145525811)

[2.1.2. Y por qué consideras que es la solución adecuada. 6](#_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. 7](#_Toc145525815)

[2.4. Auto-Rúbrica 8](#_Toc145525816)

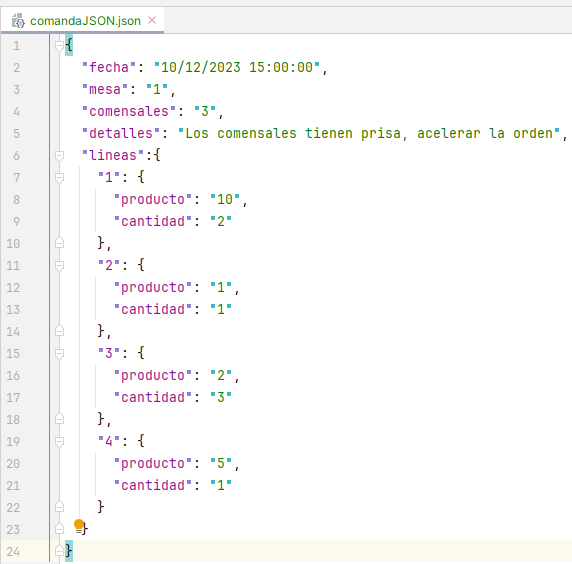
# Introducción de la actividad

Esta actividad evaluable consiste en crear una API que sirva para que una aplicación de cliente pueda conectarse al servidor del BAR JUAN.

El objetivo es que en el Bar JUAN puedan usar la aplicación tanto para las comandas como para la generación de tickets y gestión de pedidos a los proveedores.

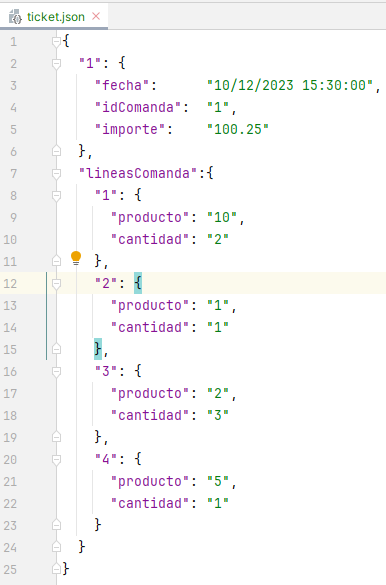
## Enunciado.

1. Usando la estructura que hemos ido viendo en la asignatura crearemos una aplicación web, que use un patrón **MVC, TWIG y Doctrine**. No es necesario usar PHP-DI.
   1. Se tendrá en cuenta si se usa un contenedor de Docker para el desarrollo. El contenedor debe tener instalado como mínimo:
      1. PHP 8 o superior.
      2. Una BB.DD. mysql.
      3. Instalado el gestor de dependencias Composer.
      4. Y cualquier otra aplicación o sistema que se considere necesario.
   2. Entregar el fichero DockerFile con la configuración del contenedor.
2. A partir de la plantilla usada en el AEV1, debes crear una pantalla inicial que sirva para que en el BAR Juan puedan acceder a ver el **stock** y realizar **pedidos** a los proveedores.
   1. En la sección de stock, tendremos a fecha del día cual es el último stock de todos los productos que tengas existencias. Para ellos, deberemos listar de la tabla ***stock*** cada uno de los ***productos*** con la última fecha y la cantidad de su stock.
   2. También podremos introducir una fecha y nos mostrará el ***stock*** de cada ***producto*** para esa fecha.
   3. Si entramos en la sección de ***pedidos***, debe entrar en un formulario que nos permita crear un ***pedido*** nuevo a un ***proveedor***. Una vez creado un pedido, nos mostrará una tabla en la que añadiremos todas las ***líneas que tenga ese pedido***. Al guardar todas las líneas, daremos por finalizado el pedido y generará por pantalla de una estructura JSON con todos los datos del pedido y las líneas del pedido.
   4. Además de esto, tendremos la opción en la pantalla anterior, en vez de generar un pedido, poder listar todos los pedidos existentes. En cada pedido debe mostrar las líneas asociadas a cada uno de los pedidos. Este listado se generará por pantalla en una estructura JSON.
3. En el Bar Juan tienen unos terminales, con una aplicación de cliente que se encarga de las gestiones de las comandas y organización de la sala, con la distribución de las mesas y los tickets.
   1. La gestión de comandas a cocina será a partir de la creación de las comandas por parte de los camareros, que se grabarán en la tabla **comandas** y ***lineascomandas***. Será el cliente quien cree un formulario para la inserción de la comanda y por lo tanto, nosotros recibiremos mediante una petición ***POST*** con los siguientes datos remitidos en una estructura JSON asociada.



* 1. De igual forma, es posible que alguna mesa decida modificar la comanda, por lo que recibiremos por ***PUT*** a la misma ruta que teníamos en el punto anterior, de nuevo la estructura JSON asociada y con el mismo formato, de forma que tendremos que actualizar la comanda actual.
  2. En ambas opciones deberemos devolver la **id de la comanda** y en la respuesta el código de estado correspondiente: **201** si se ha creado o actualizado correctamente ,**400** si algún dato no se puede procesar adecuadamente y **500** si no se ha podido procesar la petición por algún error del servidor.

1. Cada vez que desde cocina saquen una línea de la comanda, su terminal de cliente se encarga de enviar una actualización al servidor, con la línea de la comanda y la petición ***PATCH*** a una ruta exclusiva para este tipo de actualización, por ejemplo: ***/entregadaLineaComanda***,
   1. Al mismo tiempo que actualizamos la línea de la comanda, deberemos hacer una nueva entrada en la tabla de ***stock***, en la que deberemos buscar cual es el último stock de ese producto y restarle la cantidad de la línea de la comanda.
   2. Si al entregar la línea de la comanda, estuvieran todas las líneas de la comanda como entregadas, actualizaremos el estado de la comanda también.
   3. Deberemos devolver una respuesta con el código de estado correspondiente: **201** si se ha creado o actualizado correctamente ,**400** si algún dato no se puede procesar adecuadamente y **500** si no se ha podido procesar la petición por algún error del servidor.
2. Cada camarero tiene en su terminar de cliente la opción de generar ticket. Para ello, deberá siempre de estar la comanda en estado de cerrada.
   1. Para poder generar el ticket, lo primero que tendremos que hacer es comprobar el estado de la comanda, de forma que, si la comanda aún esta activa, no se podrá generar.
   2. Una vez confirmado que se puede hacer el ticket, mediante la id de la comanda deberemos generar un ticket. En nuestro servidor generar un ticket equivale únicamente a insertar en la tabla ***tickets*** un nuevo registro con los datos de la comanda. Es importante, deberemos calcular el importe total del ticket, que será igual al precio por la cantidad de cada producto de las líneas de la comanda.
   3. Una vez generado el ticket, devolveremos al cliente una estructura JSON que tendrá que contener todos los datos necesarios para que el cliente pueda sacar impreso el ticket. Se entrega un ejemplo de formato de esa estructura y en la respuesta el código de estado correspondiente: **201** si se ha creado o actualizado correctamente ,**400** si algún dato no se puede procesar adecuadamente y **500** si no se ha podido procesar la petición por algún error del servidor.



## Recursos Entregados.

Presentaciones y videos de los temas 1, 2, 3, 4, 5, 6 y 7: Material de apoyo:

* Todo lo visto en el curso hasta ahora.

Material de apoyo:

* Usar script creación BB.DD. **barjuan.sql** en formato SQL que se ha entregado junto con los recursos de la actividad.
* Hoja de rubrica para rellenar la autorúbrica y conocer los diferentes niveles de evaluación de la actividad.

## Objetivos. Rúbrica.

Mínimos (hasta 5 puntos)

* + Crear estructura de carpetas según el patrón MVC correctamente como se ha visto en los últimos ejemplos.
  + Realizar el modelado de las tablas: ***proveedores, stock, productos, pedidos y lineaspedidos***.
  + Memoria según documentación subida en Florida Oberta.
  + Realizar video explicativo de la actividad.
  + Que la ruta raíz, “/”, se muestre al iniciar el proyecto desde la carpeta public, ejecutando los requisitos que se han establecido en el punto 2 y sus subpuntos.

Suficiencia (hasta 6 puntos)

* + Cumplir todos los requisitos del punto mínimo.
  + Modelado de las tablas: ***mesa***, ***comandas y lineascomandas.***
  + Añadir la implementación del punto 3 y sus subpuntos**.**

Notoriedad (hasta 8 puntos)

* + Cumplir todos los requisitos del punto suficiencia.
  + Añadir la implementación del punto 4 y sus subpuntos.

Excelencia (hasta 10 puntos)

* + Cumplir todos los requisitos del punto notoriedad.
  + Modelado de la tabla: **ticket.**
  + Añadir la implementación del punto 5 y sus subpuntos.
  + También se valora en este punto si se ha entregado el fichero dockerFile.
  + Además del uso del repositorio GIT durante el desarrollo de la aplicación. **(El repositorio será el entregado por el profesor y no uno propio)**

# Solución de la actividad.

Esta actividad corresponde a una actividad evaluable, que nos permite comprobar si hemos entendido y afianzado conceptos básicos del MVC para PHP con un sistema de enrutado en la url del navegador y el dinamismo de las plantillas HTML con TWIG. Del mismo modo se utilizará un ORM (Doctrine) para convertir datos entre **sistemas de tipos** de un **lenguaje de programación orientado a objetos** y una **base de datos relacional**. El resultado es una **base de datos orientada a objetos** que puede usarse desde el lenguaje de programación utilizado: las **clases** son **tablas** de la base de datos y los **objetos**son**registros**. Así resulta más fácil crear y manipular tablas y datos. Finalmente la aplicación derivará en una API, en función de los parámetros y la petición recibida, devolveremos un JSON y el código de respuesta HTTP correspondiente.

# Descripción de la solución y porques.

Esta solución implementa mediante un archivo JSON las estructuras que vamos a gestionar en nuestra aplicación y a las que redijéremos si no reconocemos la ruta. También tenemos un archivo .ENV para obtener los datos de configuración de la BB.DD y otro JSON para instalar COMPOSER implantando las especificaciones de autoload PSR-4 y la instalación de TWIG y sus extensiones que vayamos a usar. La estructura de la aplicación se basa en una carpeta para la configuración: *config,* otra de acceso público: *public* y otra llamada *src* donde ubicaremos toda la lógica de nuestro MVC y el core del mismo. También encontraremos la carpeta *entity* y *repository* (antigua carpeta model) que servirá para especificar las consultas correspondientes a cada tabla de la B.B.D.D.

Finalmente, la aplicación se visualizará en el navegador tal y como se describe en el apartado 2, a través de plantillas TWIG, las cuales agilizan el dinamismo de la página debido a que son almacenadas en caché. Para los apartados 3, 4 y 5 la visualización será un JSON con los parámetros establecidos.

# Porqué has implementado esta solución.

Esta solución se ha implementado para garantizar en todo momento que el cliente vea únicamente lo que nosotros queremos que vea. Que está gestionado por el archivo Dispacher.php. Además, como estamos siguiendo un patrón MVC, se establecen la lógica por separado para los controladores, los modelos y las vistas como se me pedía en el enunciado. En el apartado 2, cada ruta corresponde a un controlador y a una acción tal y como veíamos en la primera evaluación. A partir del ejercicio 3 trabajaremos todo desde el MainController, dónde no mandará al controlador y método correspondiente en función de la ruta y el método de la petición recibida.

# Y por qué consideras que es la solución adecuada.

Considero que es la solución más adecuada según mis conocimientos, porque mantiene separados los datos de la lógica de negocio, así como el core de la propia aplicación. Se intenta reutilizar lo máximo el código y se establece como patrón de diseño un MVC. Siendo una solución completa y extensible según las necesidades que tengamos. Del mismo modo, TWIG ofrece agilidad a la hora de moverse entre los diferentes secciones de la aplicación.

# Y por qué otra solución no es más adecuada.

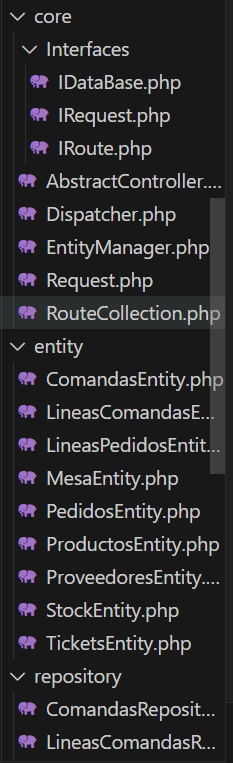
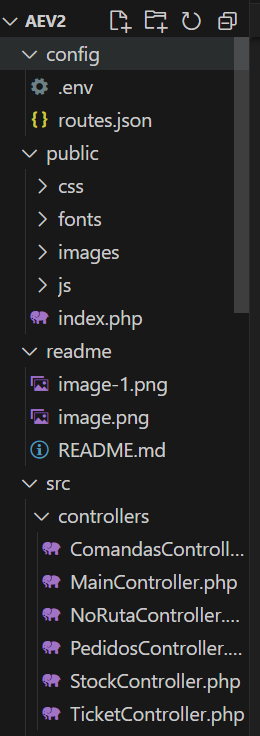
Se podría crear una mega clase y crear un Script HMTL o PHP por página y que estas nos redirijan a otras páginas, pero no separamos los datos de la lógica y estaríamos descuidando los principios SOLID.

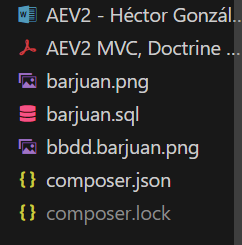
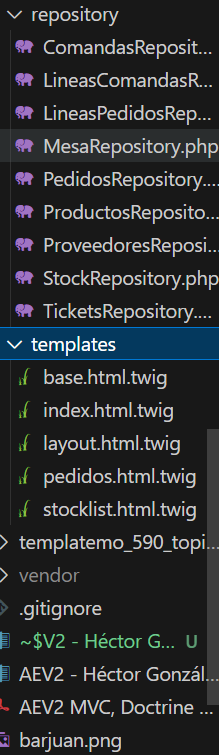
# Repositorio de Git de la actividad.

# <https://github.com/fdiaz-alonso/AEV2-HectorGonzalezMora>

# Árbol y estructura del proyecto.

La estructura del proyecto para esta actividad es:





# 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** | X |  |  |
| **Suficiencia** | X |  |  |
| **Notoriedad** | X |  |  |
| **Excelencia** | X |  |  |